UNIVERSIDAD COMPLUTENSE DE MADRID

FACULTAD DE INFORMÁTICA

DEPARTAMENTO DE ARQUITECTURA DE COMPUTADORES Y AUTOMÁTICA

TESIS DOCTORAL

Computación emergente y auto-organización aplicada al diseño de algoritmos bio-inspirados de búsqueda heurística

PRESENTADA POR

José María Benítez Escario

Directores

Juan Jiménez Castellanos
José M. Girón-Sierra

Madrid, 2015

UNIVERSIDAD COMPLUTENSE DE MADRID

FACULTAD DE INFORMÁTICA

Departamento de Arquitectura de Computadores y Automática

Computación emergente y auto-organización aplicada al diseño de algoritmos bio-inspirados de búsqueda heurística

Memoria para optar al grado de doctor presentada por

José M. Benítez Escario

Bajo la dirección de

Dr. Juan Jiménez Castellanos

Dr. José M. Girón-Sierra

Agradecimientos

En primer lugar quiero agradecer a mis directores la dedicación, el apoyo y la ayuda que me han ofrecido a lo largo del tiempo que he estado elaborando la presente tesis. A Juan por la paciencia que ha tenido a lo largo de todo este tiempo, por corregirme hasta la extenuación y guiarme en este complicado mundo que es la investigación; y sobre todo por hacerme comprender que un rechazo no es más que el primer paso del siguiente camino. A José María por compartir su experiencia conmigo, siempre dispuesto a ofrecer su ayuda para lo que fuese necesario.

A los compañeros del laboratorio Héctor, Santi, Fernando, Miriam, Lucía por hacer de un lugar de trabajo un sitio donde te apetecía ir.

A todos los miembros del Departamento de Arquitectura de Computadores y Automática y al grupo de investigación ISCAR. Con especial mención a Eva, siempre ocupada pero con tiempo para ayudar en lo que sea.

A todos los amigos que me han acompañado durante estos años. En especial a Alejandro y Manuel, al cabo del tiempo uno se da cuenta que reír, compartir,... es una pieza capital para cualquier actividad que hagamos en la vida. Nunca hay que subestimar la contribución que hace tomarse una copa en buena compañía al desarrollo de una tesis doctoral.

Y por último pero no menos importante, a mi hermano y a mi madre porque sin ellos no hubiera sido posible. Por la paciencia al escucharme y la impaciencia porque acabase. Es la mejor familia que uno pudiese desear.

Resumen

El presente trabajo de tesis doctoral se enmarca dentro de los métodos de búsqueda heurística en Inteligencia Artificial. Más concretamente se ha centrado en el diseño de algoritmos utilizando una perspectiva de computación emergente y auto-organización.

El diseño se inspira en las capacidades de auto-organización de las colonias de insectos. Esta auto-organización consiste en ajustar la conducta en función de la respuesta obtenida al realizar una acción en un determinado entorno. Este mismo esquema se ha trasladado a un algoritmo de búsqueda: las acciones serían la generación de soluciones y el entorno sería el problema que se desea resolver. De este modo se consigue que el algoritmo se auto-organice según el estado de la búsqueda.

El punto de partida ha sido un sistema multi-agente: la meta-heurística Ant Colony Optimisation. La cual ha sido modificada para aplicar un enfoque clásico de Inteligencia Artificial: búsquedas en espacios de estados. Los agentes del sistema operan de manera asíncrona. De este modo, se reduce la influencia a la que se ve sometido cada agente, lo cual se traduce en un mejor proceso de búsqueda, al reducirse el riesgo de estancarse por una pérdida de diversidad.

Además, se ha desarrollado una dinámica auto-organizativa para regular la población de agentes. Esta dinámica de población permite mantener un equilibrio en la búsqueda mediante el balance de la población de agentes tanto en tamaño como en composición. Estas técnicas de diseño permiten disminuir el número de parámetros del algoritmo.

Todo este conjunto de ideas se materializan en la implementación de un nuevo algoritmo: Ant Colony Extended, el cual ha obtenido buenos resultados en problemas de búsqueda y optimización muy diferentes tales como el problema del viajante de comercio (TSP), problemas clásicos de programación genética, y planificación-optimización de maniobras para barcos.

Palabras clave: Inteligencia Artificial, computación emergente, auto-organización, búsqueda heurística, algoritmos bio-inspirados, sistema multi-agente, ant colony optimisation, meta-heurísticas

Abstract

The present doctoral thesis is devoted to heuristic search methods in Artificial Intelligence. More specifically, it is focused on designing algorithms using a perspective of emerging computation and self-organisation.

The design is inspired by the self-organising capabilities of colonies of insects. This self-organisation adjusts the behaviour of the system, depending on the response obtained from performing an action in a certain environment. This scheme has been translated to a search algorithm: actions would be the generation of solutions and the environment would be the problem to be solved. This ensures that the algorithm is self-organised by the state of the search.

The starting point has been a multi-agent system: the meta-heuristic Ant Colony Optimisation. Which has been modified to apply a classical approach of Artificial Intelligence: solving problemas by searching a state space. System agents operate asynchronously. This reduces the influence each agent is subjected to, which results in a better search process by reducing the risk of stagnation due to a loss of diversity.

Moreover, a self-organising dynamic has been developed to regulate the agent population. This population dynamic allows holding an equilibrium of the search, balancing the population of agents in both size and composition. These techniques allow designing algorithms with fewer parameters.

These ideas lead to the implementation of a new algorithm: Ant Colony Extended, which has performed well in very different problems of search and optimisation problems, such as the traveling salesman problem (TSP), classical problems of genetic programming, and planning-optimisation of ship manoeuvres.

Keywords: Artificial Intelligence, emergent computation, self-organisation, heuristic search, bio-inspired algorithms, multi-agent system, ant colony optimisation, meta-heuristics

Índice general

Capítulo 1

Introducción

> Saber que no se sabe constituye tal vez el más difícil y delicado saber.
>
> J. Ortega y Gasset

La inteligencia y la mente humana es un tema que ha suscitado muchísimo interés a lo largo de la historia. René Descartes en el s.XVII en su famoso *Tratado del hombre* [DA90] postula el cuerpo y la mente (alma) como dos sustancias diferentes. La interacción mente-cuerpo la resolvía mediante la glándula pineal que servía de nexo.

Lo interesante de la teoría de Descartes no es su solución, sino la problemática que plantea: la interacción mente-cuerpo. El problema fundamental viene a raíz de lo que se conoce como *clausura causal del mundo físico* (causal closure): *Todos los efectos físicos únicamente pueden tener una causa física*. El cuerpo en tanto que sustancia física, no podía ser influenciado por la mente, la cual como sustancia ideal, no pertenecía al mundo físico.

Se puede ver la problemática filosófica del problema: la mente está compuesta de ideas, sustancias ideales, y mi cuerpo es una sustancia física. Entonces, ¿cómo pueden mis ideas causar mi comportamiento? O dicho de otro modo ¿cómo algo ideal puede ser causa de un efecto físico?.

El nacimiento de la computación y la Inteligencia Artificial trajo una nueva perspectiva al debate. La máquina de Turing representa una máquina física, que escribe símbolos sobre una cinta. El movimiento de dicha máquina está gobernado por reglas, asociaciones de movimiento y símbolos, dependiendo del símbolo, la máquina escribía, se movía, etc... Dependiendo del orden de los símbolos, el patrón de movimiento de la máquina cambiaba. Filosóficamente, la computación demostraba que sustancias no físicas, como la sintaxis, el orden de los símbolos, podrían ser causa de un fenómeno físico: el comportamiento de una máquina. Nace entonces la llamada *metáfora computacional*: la mente es una computadora. Esta hipótesis gozó de gran popularidad impulsada por el nacimiento de la Inteligencia Artificial a mediados del s. XX.

La Inteligencia Artificial oficialmente nace en la famosa conferencia de Dartmouth College en 1956 auspiciada por John McCarthy, Marvin Minsky, Allen Newell y Herbert Simon, considerados como los padres fundadores de la investigación en Inteligencia Artificial. La disciplina tiene su germen en un artículo

publicado por Alan Turing [Tur50], donde plantea la cuestión de si una máquina puede pensar.

En el contexto histórico en que nace la Inteligencia Artificial existe otro hecho fundamental: el nacimiento del cognitivismo. El cognitivismo es una corriente ideológica que nace entre 1950 y 1960 como reacción al conductivismo, el cual postulaba la mente humana como una caja negra inaccesible, donde el comportamiento puede explicarse recurriendo únicamente a asociaciones estímulo-respuesta. En contraste, el cognitivismo afirma que el comportamiento debe atender a los procesos mentales: procesamiento de información. El individuo percibe información mediante los sentidos y la procesa, el resultado de ese procesamiento da lugar a diferentes conductas. El cognitivismo va ligado a la idea de computación en tanto que la computación constituye la definición formal, matemática, de procesamiento de la información.

Definir la mente humana como un procesamiento de información, no como una caja negra, hace posible trasladar ese procesamiento a otro sistema, a un ordenador. De tal modo que si un ordenador procesa la información como lo hace la mente humana, al ser ésta inteligente, el ordenador también sería inteligente. De este modo las primeras investigaciones en Inteligencia Artificial tienen una profunda raíz teórica, textos como "Human Problem Solving" [NS+72] de Newell y Simon profundizan en modelos o hipótesis de cómo es un proceso cognitivo mediante modelos computacionales.

La Inteligencia Artificial consiguió grandes avances en distintos campos, pero la meta principal se asumió casi como imposible: no fue capaz de recrear la inteligencia humana. La imposibilidad de la Inteligencia Artificial trajo como consecuencia el declive de la metáfora computacional, en ciertos aspectos la mente es parecida a un computador, se acepta que procesa información, pero no es un computador. En su lugar apareció la *Neurociencia*: el estudio del cerebro. Esta nueva perspectiva al ser de base biológica, abrió la puerta al estudio del cerebro como una adaptación evolutiva.

La aproximación de la Neurociencia es la vigente a día de hoy pero lleva consigo un problema: mientras se profundiza en la compresión del cerebro, apenas se obtienen respuestas acerca de la mente humana. Términos como "inteligencia", "conciencia", "el yo", etc... siguen siendo problemáticos.

La Neurociencia unida a las llamadas ciencias de la complejidad, el estudio de sistemas complejos, trajo una nueva hipótesis: la mente como fenómeno emergente de la dinámica cerebral. La novedad de esta teoría radica en que hasta ahora siempre se había buscado una explicación explícita de la mente y por ende de la inteligencia, una definición concreta. Esta aproximación propone la mente como fenómeno. La idea es, por lo menos, sugerente.

Igual que ocurre con la mente humana, la misma aproximación se puede aplicar a la inteligencia artificial. No hay ninguna razón para que la inteligencia tenga que ser diseñada explícitamente mediante un conjunto de procedimientos o instrucciones. Como se puede encontrar en algunos textos de R. Brooks [Bro90, Bro91, Bro99], la inteligencia puede ser una propiedad de un sistema computacional, en el sentido que el sistema se comporte de manera inteligente sin necesidad de que en su programación haya ninguna mención explícita a la inteligencia. La inteligencia resulta una propiedad emergente.

En esta tesis lo que se propone es diseñar un algoritmo de búsqueda capaz de resolver problemas de una manera inteligente, donde la inteligencia del algoritmo surja de la dinámica propia del algoritmo, es decir, sea de carácter emergente.

1.1. Comportamientos emergentes y auto-organización

La primera cuestión consiste en definir que es un "comportamiento emergente", una definición adecuada la podemos encontrar en el texto de Steven Johnson [Joh02]:

> A higher-level pattern arising out of parallel complex interactions between local agents.

El término *emergente* se aplica a aquellos sistema que exhiben patrones de comportamiento causados por las interacciones locales de sus agentes. Es decir, dicho patrón no es consecuencia de las capacidades de cada agente, sino de la interacción entre ellas.

Cuando hablamos de *patrón*, implícitamente estamos introduciendo un sistema que evoluciona en el tiempo. Si en instantes sucesivos de tiempo el "aspecto" del sistema permanece estable, no cambia, entonces dicho aspecto constituye un patrón: una repetición temporal.

Ejemplo típicos los encontramos en la evolución, a lo largo del tiempo, una especie puede evolucionar en otra. Ambas especies constituyen un patrón de una forma de vida, mientras que en el proceso lo que se dan son estados intermedios –transiciones– donde se va concretando la morfología de la nueva especie.

Las propiedades emergentes constituyen un área multi-disciplinar, dentro de la cual podemos encontrar multitud de estudios de diferentes autores [Hol75, L+89, For90, Kau93, Kau95, Mit09].

En nuestro caso nos vamos a centrar en lo que se conoce como *computación emergente* que consiste en el estudio de propiedades emergentes en sistemas computacionales.

Computación emergente

El inicio de lo que se conoce como *Computación Emergente* lo podemos fechar "oficialmente" en una conferencia del año 1990 que tuvo lugar en "Los Alamos National Laboratory". La conferencia, organizada por Stephanie Forrester con el titulo de "Emergent Computation" [For91], tenía como objetivo dar una definición al área de estudio de la emergencia dentro de las ciencias computacionales.

La propia Forrester [For90] nos da una definición de qué constituye la *computación emergente*:

> (i) A collection of agents, each following explicit instructions
>
> (ii) Interactions among the agents (according to their instructions), which form implicit global patterns at the macroscopic level, i.e. epiphenomena
>
> (iii) A natural interpretation of the epiphenomena as computations

Lo que indica Forrester es básicamente la misma idea de emergencia pero trasladada a un sistema computacional: los agentes tienes reglas explícitas programadas e interactuan entre ellos. De dicha interacción se puede observar un patrón global de comportamiento entre los agentes. Dicho patrón es un fenómeno secundario derivado de uno primario: la interacción entre los agentes.

La pregunta clave es ¿por qué podemos considerar este segundo fenómeno, el patrón global, como computación? La respuesta proviene de la teoría de la

computación. Como señala Forrester, de acuerdo con la tesis de Church-Turing, una máquina de Turing puede implementar cualquier posible sistema computacional y ejecutar cualquier conjunto de instrucciones. Por tanto las misma reglas que definen un fenómeno de computación emergente pueden ser definidas como una máquina de Turing, y el epifenómeno que denominamos emergente, no sería más que el cómputo realizado por dicha máquina de Turing.

La computación emergente no tiene propiedades mágicas, sigue siendo computación. La diferencia es que el sistema está diseñado de otra manera, la computación no se expresa de una manera explícita mediante un conjunto de reglas, sino que surge, es consecuencia, de la interacción de conjuntos de instrucciones más sencillas (agentes) entre sí. El gran avance de la computación emergente es que posibilita el diseño de sistemas computacionales complejos a partir de otros más sencillos.

Para ver en qué consiste la computación emergente vamos a introducir los autómatas celulares (CA) [Lan90, Wol02] que constituyen el ejemplo típico de computación emergente.

Autómatas Celulares

Formalmente se definen como una matriz de dimensión $n \times m$ donde en cada celda de dicha matriz reside un autómata. Cada autómata recibe como entradas los estados de los autómatas adyacentes, dicha región se conoce como el *vecindario* del autómata. El estado de cada autómata es función de sus vecinos.

Por ejemplo, un autómata celular de dos dimensiones con un vecindario de 8 vecinos se representa de la siguiente manera:

a_0	a_1	a_2
a_3	x	a_4
a_5	a_6	a_7

la función de transición(δ) que dictamina el estado de x viene dada por los estados de sus vecinos: $\delta(x) = f(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7)$. Dependiendo de la definición de δ podemos obtener un autómata u otro.

Típicamente los autómatas celulares se utilizan para el estudio de sistemas físicos, ya que permiten estudiar los comportamientos de sistemas no lineales. Por ejemplo, un autómata celular muy conocido es el denominado "majority rule". Cada autómata puede tener dos estados 1, voto favorable, o 0 voto desfavorable. Primeramente, se inicializa al azar cada autómata, después se procede con la evolución de la siguiente manera:

- $estado = 0$ si en el vecindario del autómata la opción mayoritaria es 0 : (n^o 0 $>n^o$ 1)
- $estado = 1$ si en el vecindario del autómata la opción mayoritaria es 1 : (n^o 1 $>n^o$ 0)
- En caso de empate (n^o 0 = n^o 1) el estado permanece inalterado.

En la figura 1.1 podemos ver un ejemplo de este autómata con un vecindario de 8 vecinos para cada autómata. Inicialmente, iteración 1, cada autómata representa un individuo y sus creencias: voto favorable o desfavorable. Pero las creencias de dicho individuo se ven influenciadas por sus conocidos (vecindario), de tal modo

que si éstos muestran una opinión clara (mayoría) entonces el individuo se alinea con dicha opción. El autómata celular simula cómo la alineación de los individuos se propaga por la sociedad ya que un conocido mío, es conocido de otro. De tal modo que mi opinión influye directamente en los individuos más cercanos e indirectamente en los más lejanos.

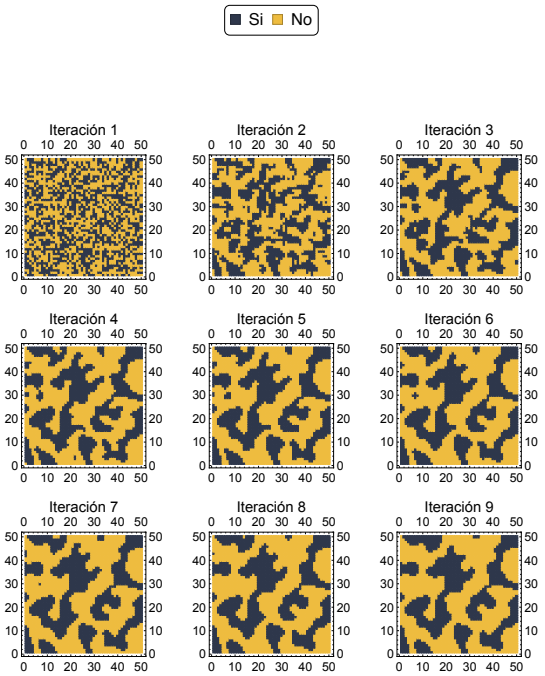


Figura 1.1: Evolución de un autómata celular "majority rule" de tamaño 50×50, con un vecindario de 8 vecinos.

.

Si observamos el autómata, en el estado inicial no hay más que caos, cada individuo tiene su opinión y ésta únicamente responde a sus creencias internas que se modelan por azar. Ahora bien, al interactuar los individuos, los autómatas, se alinean los unos con otros y van creando patrones de opinión. En la evolución se puede observar cómo aquellos grupos reducidos inmersos en una corriente de opinión contraria tienden a desaparecer.

Finalmente, el autómata celular pone de relieve como existe una tendencia, una convergencia, hacia la formación de patrones: grandes grupos estables con una opinión compartida. Estos patrones constituyen el fenómeno emergente ya que no están programados en el conjunto de instrucciones que definen un autómata, sino que el orden que exhiben los autómatas en su conjunto es producto de la relación entre ellos.

Uno de los pioneros en el estudio de los autómatas celulares fue S. Wolfram [Wol02, Fla98], el cual estableció diferentes clases:

Clase I: la evolución es estable y homogénea, existe convergencia hacia un único estado.

Clase II: la evolución converge a un patrón de estados, trayectoria, que es estable y periódica.

Clase III: la evolución es inestable, no existe convergencia hacia ningún tipo de patrón. Esta falta de orden se denomina *caos*. Igualmente un sistema desordenado se dice que es caótico[1].

Clase IV: la evolución converge a comportamientos complejos donde "conviven" el orden y el caos. Es decir, coexisten tanto regiones ordenadas, en forma de patrones locales, como regiones desordenadas, sin presencia de ningún patrón.

En la figura 1.2 se ilustran las distintas clases. Se trata de autómatas unidimensionales: vectores. En la figura se muestra cómo evoluciona en el tiempo dicho autómata, el eje de ordenadas representa el tiempo (iteraciones). Como se puede ver, la clase 1, converge a un único estado, la clase 2 tiene un patrón repetitivo, la clase 3 es caótica, no muestra ningún patrón; y la clase 4 es compleja: los patrones son locales, no se dan a lo largo de todo el autómata sino que se mueven en el espacio – a lo largo del vector– y del tiempo –de una iteración a otra–.

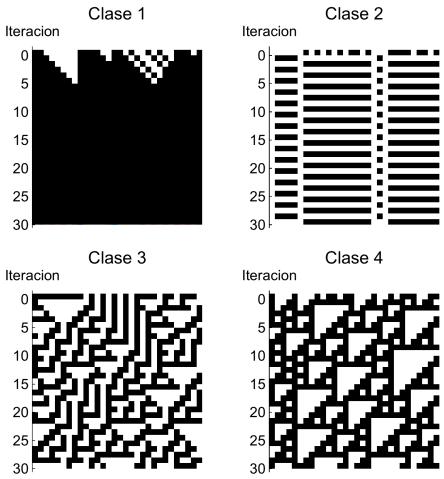


Figura 1.2: Ejemplos de distintas clases de autómata unidimensionales según el criterio establecido por Wolfram.

La clase IV de autómatas de Wolfram son un ejemplo de lo que se denomina "computation at the edge of chaos", el término fue acuñado por C. Langton uno de los fundadores de lo que se conoce como "Artificial Life" [L+89] que consiste en el estudio y simulación de los sistemas biológicos mediante modelos computacionales. En un artículo publicado en 1990 [Lan90], Langton sugiere que un sistema

[1] Aunque guarda relación, el término caótico no se utiliza en el mismo sentido que en la teoría del caos, únicamente para indicar la ausencia de patrones ordenados.

físico con capacidad para realizar cómputo, encuentra las condiciones óptimas de procesamiento cuando se encuentra cercano, en las vecindades, del cambio de fase (phase transition). En estas condiciones es cuando aparecen los comportamientos complejos.

En este contexto, *cambio de fase* [SG08] hace referencia a sistemas complejos y ha de entenderse como el cambio de un *estado ordenado* a un *estado desordenado*. Hablamos de estado ordenado cuando la configuración del sistema presenta un patrón claro, es decir, un orden. Por ejemplo, en la figura 1.2 los autómatas de clase I son sistemas ordenados. A medida que disminuye el orden en el sistema, vamos avanzando hacia el cambio de estado. Los autómatas de clase II siguen siendo sistemas ordenados, pero menos que los de clase I. Finalmente alcanzamos lo que se denomina estado desordenado (caótico), donde no se vislumbra ningún patrón que predomine en el sistema, por ejemplo los autómatas de clase III.

Como se puede intuir existe un gradación desde más ordenado hasta que el sistema entra en un estado caótico. Langton hace referencia a los últimos estadios de un sistema ordenado, cuando está a punto de cambiar de fase, es decir, pequeños fragmentos de orden en un mar caótico, como por ejemplo los autómatas de clase IV. Es lo que se denomina "edge of chaos" o frontera del caos. En estos estadios el sistema presenta unas características "especiales" puesto que ni está ordenado, ni desordenado, digamos que está cambiando, está en transición.

Langton sugiere que en ese estadio de transición es cuando el sistema presenta la máxima flexibilidad manteniendo un mínimo de coherencia, lo cual maximiza las capacidades de cómputo del sistema, es decir, el procesamiento de información. En computación la información debe entenderse como la variabilidad de un conjunto de símbolos (véase el apéndice A). Igualmente, cuando se habla de procesamiento de la información no debe entenderse como comprensión u entendimiento. El procesamiento debe entenderse como la capacidad de un sistema para producir información según la información que recibe.

Para comprender la idea de Langton vamos a ver un ejemplo tomado de [SG08], una simulación de propagación de fuego en un bosque. La simulación la realizamos con autómatas de dos dimensiones. Partimos de una probabilidad h que indica la probabilidad de que una celda del autómata contenga un árbol o esté vacía. A continuación, inicializamos el autómata y "prendemos fuego" al extremo inferior del autómata. La figura 1.3 muestra los estados iniciales de la simulación para distintos valores de h. La función de transición es muy sencilla, utilizamos vecindarios de 8 autómatas, de tal modo que si el autómata es un árbol y hay fuego en su vecindario, éste se propaga y el árbol se quema. Es decir, un autómata árbol pasa al estado fuego si existe fuego en su vecindario.

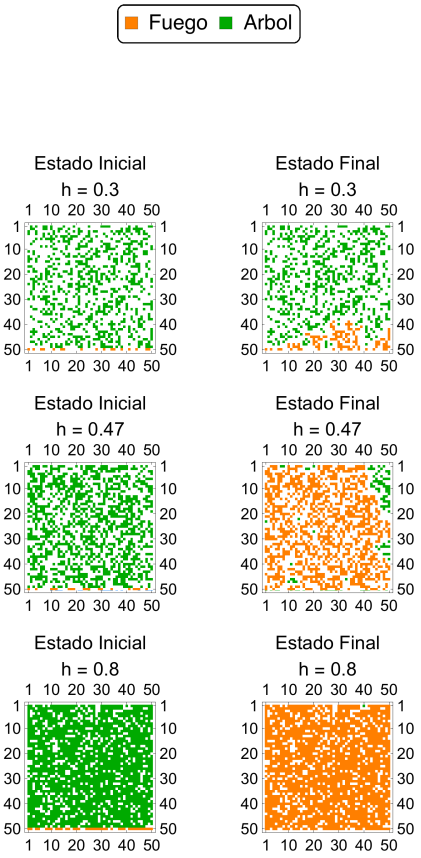


Figura 1.3: Simulación de propagación de fuego en un bosque, el término h es una medida de la densidad del bosque.

.

Como se ve en la figura 1.3 si la probabilidad h es baja, no hay apenas densidad de árboles y el fuego no se propaga, el estado final contiene casi el bosque intacto. En cambio, si la densidad es alta, el fuego se propaga y todo el bosque arde. Si la probabilidad es intermedia, el resultado varía, existirán zonas que se salven de la quema y otras que aparezcan chamuscadas.

Lo interesante de esta simulación viene a continuación: ¿cuánto tarda el fuego en estabilizarse en función de la densidad del bosque? consideraremos que el fuego se estabiliza cuando no puede propagarse más, bien porque todos los árboles estén ardiendo, o bien porque no pueda alcanzar a más árboles. El tiempo de propagación lo podemos medir en el número de iteraciones que tarda el autómata celular en converger.

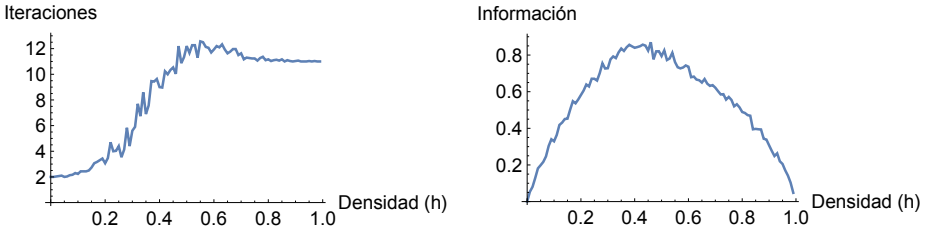


Figura 1.4: Nº de iteraciones que tarda en estabilizarse la simulación de propagación del fuego en un bosque y contenido de información del estado final.

.

En la figura 1.4 mostramos los resultados de realizar la simulación. Podemos observar que a medida que aumentamos la densidad aumenta el tiempo requerido para alcanzar un estado estable, ahora bien, el tiempo máximo no se da cuando la densidad es máxima, sino cuando la densidad está próxima a 0,6. La razón es muy sencilla, a mayor densidad, mayor número de árboles y más posibilidades para que el fuego se propague de un autómata a otro. Si la densidad decae, todavía es posible que se alcancen los árboles más alejados del borde inferior. Pero el fuego no puede seguir muchas rutas, únicamente unas pocas, con lo cual tarda más en propagarse. Si además nos fijamos en el estado final que es estable, más concretamente en su contenido de información –la cantidad de información necesaria para representar dicho estado–, podemos ver que la información es máxima cuando coinciden los patrones de árboles y fuego. Por el contrario en los extremos no existe esa diversidad o hay árboles o hay fuego, pero no ambos.

Esta tesis no tiene como objetivo el estudio de la computación emergente en sí misma, sino como principio de diseño aplicado al campo de algoritmos meta-heurísticos. A continuación vamos a introducir en qué consisten los algoritmos meta-heurísticos y cómo los podemos relacionar con lo que hemos visto en los autómatas celulares.

Meta-Heurísticas

El término meta-heurística aparece por primera vez en 1986, Glover lo utiliza para describir la búsqueda tabú [Glo86] como un procedimiento que superpone a una heurística determinada. Literalmente, "meta-heurística" significa "más allá de la heurística". En general, una meta-heurística consiste en estrategias de diseño para procedimientos heurísticos, aproximados, aplicados a la resolución de problemas complejos. Una revisión completa de diversos algoritmos meta-heurísticos la podemos encontrar en [GK03].

Las meta-heurísticas las podemos clasificar en diferentes estrategias, véase por ejemplo [BM03]. Para este trabajo resultan particularmente interesante los siguientes tipos:

- *Constructivas*, basadas en la selección individual de las partes que componen una solución. Ejemplo típico los algoritmos del tipo Ant Colony Optimization [DS04].

- *De búsqueda*, basada en la transformación de una solución en otra mediante operadores de transformación. Ejemplo típico los algoritmos como búsqueda Tabú [Glo86] y el templado o recocido simulado [KGV+83].

- *Evolutivas*, basadas en la evolución de conjuntos de soluciones. Ejemplo típico los algoritmos genéticos [Gol06].

Un procedimiento meta-heurístico lo podemos entender como un método iterativo de búsqueda de soluciones. En cada iteración se obtienen nuevas soluciones de acuerdo con la estrategia adoptada, bien por composición –estrategia constructiva– o por combinación –estrategia evolutiva–.

La parte crítica del diseño en un algoritmo meta-heurístico la exponen claramente Blum y Roli [BR03]:

> In short we could say that metaheuristics are high level strategies for exploring search spaces by using different methods. Of great importance thereby is that a dynamic balance is given between diversification and intensification. The term diversification generally refers to the exploration of the search space, whereas the term intensification refers to the exploitation of the accumulated search experience.

La búsqueda debe mantener un balance entre exploración (diversification): captación de información mediante la generación de nuevas soluciones, o explotación (intesification): uso de la información disponible mediante la generación de soluciones similares a las actuales. Es decir, el conjunto de soluciones representa diferentes áreas del espacio de soluciones. Un algoritmo puede generar una solución perteneciente a un área ya conocida. En este caso, el algoritmo no obtiene mucha información; pero, si la zona era prometedora, puede obtener una solución de calidad. Por el contrario, puede tratar de generar una solución de otra área hasta ahora desconocida, aumentando la información acerca del espacio de búsqueda.

La capacidad de un algoritmo es limitada, esto es lo que hace tan crítico el proceso. Si el algoritmo opta por explotar la información conocida, pierde información, ya que va a obtener un mayor número de soluciones de una misma zona. Como las nuevas soluciones sustituyen a las antiguas, una excesiva explotación degenera en una pérdida de diversidad en el proceso de búsqueda. Igualmente, si el algoritmo explora la búsqueda se dispersa; se conocen muchas zonas del espacio de búsqueda pero apenas se tiene información de ellas, no se sabe cual es mejor y por tanto en cual conviene centrarse para buscar soluciones de mejor calidad.

Este balance de la búsqueda lo podemos implementar explícitamente, mediante mecanismos de control más o menos complejos. Pero también podemos recurrir a la computación emergente. El mecanismo de control no sería explícito, sino una capacidad emergente del sistema. Como señala Forrester [For90], diseñar un sistema flexible puede ser imposible mediante una definición explícita de los procedimientos, pero dicha flexibilidad puede aparecer en un nivel emergente, producto de las interacciones entre el entorno y los componentes del sistema.

En este punto es donde tiene relación lo dicho anteriormente sobre los autómatas celulares, porque el contenido de información de un autómata guarda relación con el orden que exhibe dicho autómata. A medida que el autómata se desordena, aparecen más patrones, aumenta el contenido de información del autómata; y viceversa, cuando el autómata converge a un único patrón, la información decae.

Los autómata son entidades computacionales dirigidas por información, donde dicha información proviene de los estados de los autómata adyacentes. Cuando la información puede circular libremente por toda la red de autómatas, se tiende a la formación de patrones globales. Si la información puede circular parcialmente aparecen patrones locales. Es la simulación del modelo de propagación de fuego en el bosque, figura 1.3. Es decir, el orden en una autómata celular está influido por la conectividad que se de entre los autómatas.

Como el objetivo es diseñar un algoritmo meta-heurístico, necesitamos una entidad computacional más compleja que un autómata. Supongamos que disponemos de agentes. Al igual que los autómatas, los agentes son entidades computacionales dirigidas por información pero con capacidad de búsqueda. La información no determina el estado del agente, sino la solución que éste es capaz de construir.

Al igual que ocurre con los autómatas, los agentes comparten información. Si la conectividad entre los agentes es baja, cada uno funcionará aisladamente y el sistema no será eficaz. Por el contrario, si la conectividad es muy alta, los agentes convergerán a una misma información; producirán el mismo tipo de soluciones, resultando en una baja diversidad en la búsqueda. Lo que interesa es que los agentes presenten una conectividad intermedia, que permita la formación de grupos. Agentes próximos entre sí tienden a trabajar con una misma información, explorando un determinado conjunto de soluciones; mientras que agentes más alejados tienden a trabajar con otra información, en otro grupo. Es decir, los agentes se agrupan y cada grupo buscaría en un área de soluciones distinto, de este modo conseguimos un balance entre exploración: se cubren varias zonas de búsqueda, y explotación: se adquiere un buen conocimiento de las soluciones presentes en cada zona.

Auto-organización

La idea de agrupar esfuerzos de búsqueda por zonas no es ni mucho menos original, la originalidad radica en que esa agrupación se haga de manera emergente. Al ser una propiedad emergente se flexibiliza enormemente, ya que no están definidas cuantas zonas deben cubrirse ni el número de agentes presente en cada una de ellas. Será una forma de organización que surja del propio proceso de búsqueda en sí mismo, una forma de *auto-organización*.

La auto-organización introduce en el contexto de las capacidades emergentes de un sistema la noción de *entorno*. El entorno no forma parte del sistema, es donde se encuentra el sistema. Cuando introducimos el entorno aparece vinculado a la noción de *tarea*, *actividad*, *objetivo*. Podemos definir la auto-organización como la capacidad de un sistema para ordenarse de acuerdo con alguna finalidad en un determinado entorno. Los ejemplos más comunes de auto-organización refieren a sistemas biológicos, véase por ejemplo [CDF+03].

En los ejemplos de los autómatas celulares, la noción de entorno no existía a nivel de sistema, para cada autómata, el vecindario consistía en su entorno. Pero la red, la matriz que contiene todos los autómatas no tenía entorno. Por tanto, la evolución del autómata celular venía determinada por el estado inicial del autómata y las reglas. En la auto-organización la evolución del sistema depende de las reglas, los estados de los individuos que componen dicho sistema y del entorno.

Podemos tomar por ejemplo el vuelo en formación de los pájaros. Este comportamiento se puede simular de una manera sencilla utilizando dos simples reglas:

acercarse a las aves más alejadas y alejarse de las aves más cercanas. Siguiendo estas dos reglas las aves se auto-organizan en una formación conjunta. Si aparece un depredador en escena, las aves igualmente tratan de agruparse, pero teniendo en cuenta las nuevas condiciones del entorno: el depredador, del cual huyen. La figura 1.5 muestra una sencilla simulación de este ejemplo, como se puede ver el resultado cambia. Las reglas son las mismas en ambos casos, las condiciones iniciales de las aves también; lo único que varía es el entorno: que haya o no depredador. Esto es suficiente para que las aves muestren otra organización.

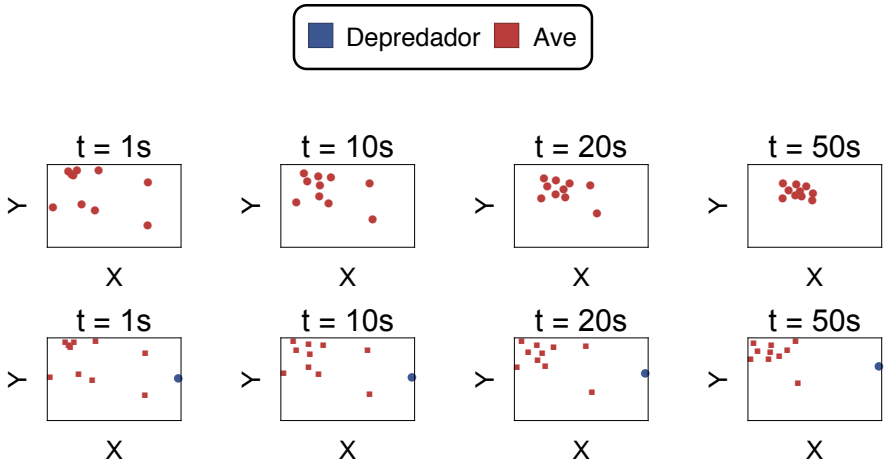


Figura 1.5: Formación de una bandada de aves en presencia o ausencia de un depredador.

En un algoritmo meta-heurístico, el entorno es el problema que se está resolviendo, el estado de la búsqueda. La auto-organización nos permite diseñar comportamientos emergentes que ordenen el sistema de acuerdo con el entorno, con el proceso de búsqueda en sí mismo. En función de la búsqueda el sistema puede organizarse para incrementar la exploración de nuevas soluciones o bien para intensificar la generación de soluciones en función de la información disponible. De este modo es posible diseñar una dinámica que permita un orden emergente con una finalidad: mantener el equilibrio en la búsqueda.

En el resto de la tesis presentaremos los pasos de diseño que muestran como incluir propiedades emergentes y auto-organizativas en el contexto de un algoritmo meta-heurístico.

1.2. Objetivos

De acuerdo con lo expresado anteriormente, podemos concretar los siguientes objetivos de investigación:

i) Definir un sistema multi-agente básico que sirva de marco de trabajo para la implementación de un algoritmo de búsqueda.

ii) Estudio de la toma de decisiones colectivas en las colonias de insectos como modelos de procesos emergentes que permitan incrementar la eficiencia en la búsqueda.

Una vez realizado el estudio se procederá de la siguiente manera:

- Las conclusiones extraídas de este estudio se utilizarán como guía de diseño de un prototipo: un algoritmo de búsqueda funcional.
- Se utilizará el prototipo para realizar un análisis experimental de cara a confirmar la viabilidad de las técnicas utilizadas.

iii) Estudio de la procesos de auto-organización de las colonias de hormigas como modelo de un esquema de control. Dicho esquema permitiría al sistema multi-agente controlar por sí mismo el proceso de búsqueda.

Una vez realizado el estudio se procederá de la siguiente manera:

- Se completará el prototipo diseñado para el objetivo anterior con el fin de incluir dinámicas auto-organizativas de acuerdo con las conclusiones extraídas del estudio realizado.
- Posteriormente, se realizará un análisis experimental de cara a confirmar la viabilidad de las técnicas utilizadas.

iv) Utilizando los desarrollos realizados con anterioridad, se definirá un algoritmo completo. Este algoritmo se aplicará a diferentes problemas con el fin de comprobar que el esquema de diseño seguido es eficaz.

De acuerdo con los objetivos definidos, la organización del documento será la siguiente:

En el capítulo 2 introduciremos la metodología Ant Colony Optimization y las búsquedas en espacios de estados. Estos dos elementos nos servirán para definir el contexto algorítmico que utilizaremos para el diseño.

En el capítulo 3 se introducen dos sistemas biológicos: las colonias de abejas y las colonias de hormigas. Tomando como modelo los procesos de toma de decisiones colectivas en estos dos sistemas, diseñaremos un prototipo de algoritmo.

En el capítulo 4 se finaliza el prototipo del algoritmo con la inclusión de una dinámica de población, nuevamente inspirada en los comportamientos auto-organizados de una colonia de hormigas. La utilizaremos para completar las capacidades del prototipo anterior.

El capítulo 5 contiene los estudios experimentales que se han llevado a cabo aplicando el algoritmo a la resolución de diferentes problemas: bancos de pruebas y una aplicación real.

El capítulo 6 presenta las conclusiones de la investigación, así como la líneas de trabajo futuras.

En las publicaciones [EJGS11, EJGS15, EJGS12, EJGS13, EJG10b, EJG10a, EJG09], se pueden consultar más detalles acerca de la investigación presentada en esta tesis.

Capítulo 2

Inteligencia Artificial Distribuida

¿Cómo puede surgir la inteligencia de algo no inteligente? Para hallar la respuesta, demostraremos que es posible construir una mente a partir de muchas partes pequeñas que entre sí no la poseen.
Llamaré "sociedad de la mente" a este modelo, según el cual cada mente está formada por numerosos procesos más pequeños. Daremos a estos procesos el nombre de *agentes*. Por sí solo, cada agente no es capaz de realizar más que alguna cosa sencilla que no requiere en absoluto poseer mente ni pensamiento.

M. Minsky "La sociedad de la Mente"

Como se vió en la introducción, la computación emergente requiere de un sistema donde las partes se relacionan entre sí. En computación una forma de definir esta clase de sistemas son los denominados *sistemas multi-agentes*, los cuales se enmarcan dentro de lo que se denomina Inteligencia Artificial Distribuida (DAI) [DMRP10, Wei99, SLB09, Min88, WJK00, ZJW03]. La DAI engloba sistemas con capacidades inteligentes, donde estas capacidades no están centralizadas, sino distribuidas entre las partes que componen el sistema. Esta distribución puede ser explícita, donde cada parte o agente del sistema posea alguna capacidad inteligente. Pero también pueden ser emergentes, siendo la inteligencia una propiedad global resultado de la interacción de los agentes.

En el contexto de este trabajo un sistema multi-agente debe entenderse como una técnica de diseño. En vez de realizar un sistema excesivamente complicado, se diseñan partes más sencillas que se relacionan entre sí: en el caso de los agentes se diseñan procesos computacionales sencillos y se coordinan entre ellos. Véase como ejemplo los estudios de coordinación para el movimiento de caracteres realizados por C. Reynolds [Rey00, Rey99].

Además de facilitar el diseño, los sistemas multi-agentes se usan frecuentemente para modelar sistemas complejos. En muchos casos no es posible describir matemáticamente el comportamiento de este tipo de sistemas, por lo que se recurre a estudios de simulación computacional. Por ejemplo, estudios de simulación social [GC95, MP09, Gil95, GD94, Axe97].

Dentro de las meta-heurísticas los algoritmos de hormigas: Ant Colony Optimisation (ACO), son ya de por sí sistemas multi-agentes. La metodología ACO forma

parte de lo que se denomina *swarm intelligence*, que significa literalmente "inteligencia de enjambre". El término es relativamente nuevo, aparece por primera vez en 1989 [BW93] pero su uso no se ha extendido hasta hace poco. El término hace referencia al comportamiento colectivo que muestran los sistemas descentralizados y auto-organizados. Realmente la "swarm intelligence" no es muy diferente de lo que antes se denominaba DAI, su particularidad es que típicamente está inspirada en modelos biológicos.

El estudio de los sistemas multi-agentes en el contexto de la meta-heurística ACO constituirá el primer paso en nuestra investigación.

2.1. Ant Colony Optimization

La forma más sencilla de sistema multi-agente es aquél que simplemente consta de una población de individuos idénticos. Por ejemplo, en la figura 2.1 se representa un agente que resuelve problemas. El término "agente" tiene multitud de acepciones. En el contexto de este estudio agente debe entenderse como una *entidad computacional autónoma* [Wei99]: entidades que existen en forma de programas que se ejecutan en algún dispositivo computacional.

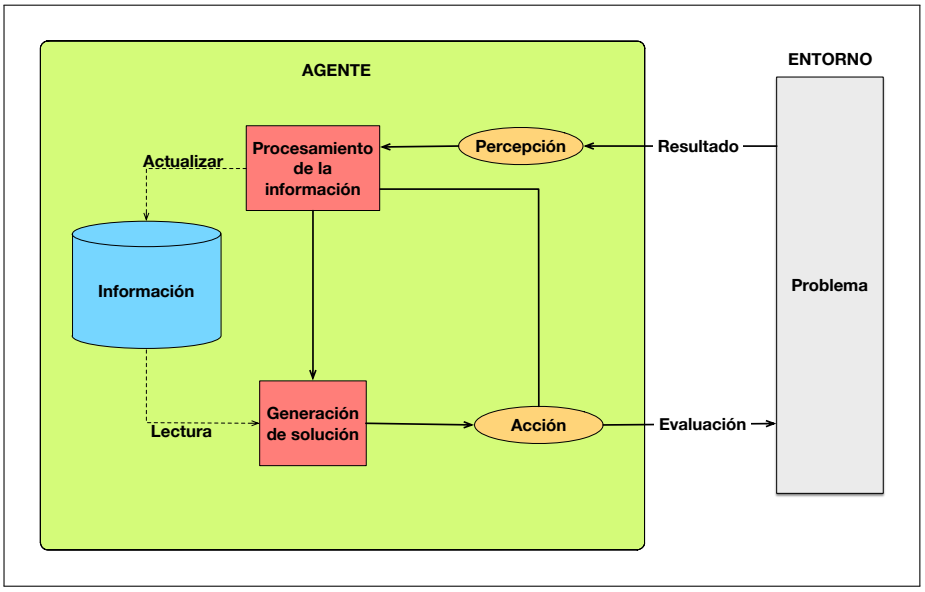


Figura 2.1: Esquema sencillo de un agente que busca soluciones a un problema.

Volviendo a la figura 2.1, como se observa, el agente es autónomo, es un bucle que genera soluciones, las cuales son evaluadas frente al problema dado y en función de esa evaluación se actualiza la información disponible. El proceso entero podría describirse de la siguiente manera:

1. Utilizando la información disponible, el agente genera una solución al problema.

2. Una vez que ha finalizado la búsqueda, se evalúa la solución obtenida.

3. El resultado de dicha evaluación, junto con la solución generada se utiliza para actualizar la información disponible.

4. Una vez que se ha actualizado la información se procede a buscar de nuevo.

Si disponemos de más de un agente como el de la figura 2.1, estamos dando un paso hacia un *Sistema Multi-agente*. El paso siguiente consiste en definir cómo se relacionan los agentes entre sí.

M. Dorigo cuando introduce por primera vez los algoritmos de hormigas con la publicación del Ant System (AS) [DMC96], no utiliza reglas explícitas para definir la coordinación de los agentes, los relaciona a través de la información. Los agentes son entidades computacionales dirigidas por información, por tanto, si la información es compartida entre un grupo de agentes, se establece un vínculo entre ellos a través de la propias información. La idea de Dorigo se basa en el concepto de "stigmergy".

El término *stigmergy* fue acuñado por el francés Pierre-Paul Grassé en 1959 para referirse al comportamiento exhibido por las termitas [Gra59]. Dorigo lo utiliza en términos de comunicación: "indirect communication through enviroment". La información que guía un agente es en principio privada, está incluida en la propia especificación del agente. Dorigo la extrae del contexto del agente y la define de forma externa. Al hacerla externa, ya no forma parte del agente sino que se convierte en entorno. A la vez, el acceso a esta información puede ser realizado por diferentes agentes al mismo tiempo, convirtiéndola en *información pública*: disponible para todos los agentes que conforman el sistema. De este modo se define un intercambio de información entre los agentes, pero de manera indirecta, mediante la modificación por parte de un agente de la información pública.

Este esquema desarrollado por Dorigo se representa en la figura 2.2. Los agentes siguen disponiendo de una información privada, la cual es necesaria para almacenar la solución generada; pero la información que guía el proceso de generación de soluciones proviene de una estructura de datos pública, compartida. Al finalizar el proceso de generación, los agentes actualizan el contenido de la información pública a partir de la solución generada, es decir, según el contenido de su información privada.

Al compartir la información es posible que los agentes se coordinen de manera espontánea, ya que la información generada por un agente puede dirigir el comportamiento de los demás agentes.

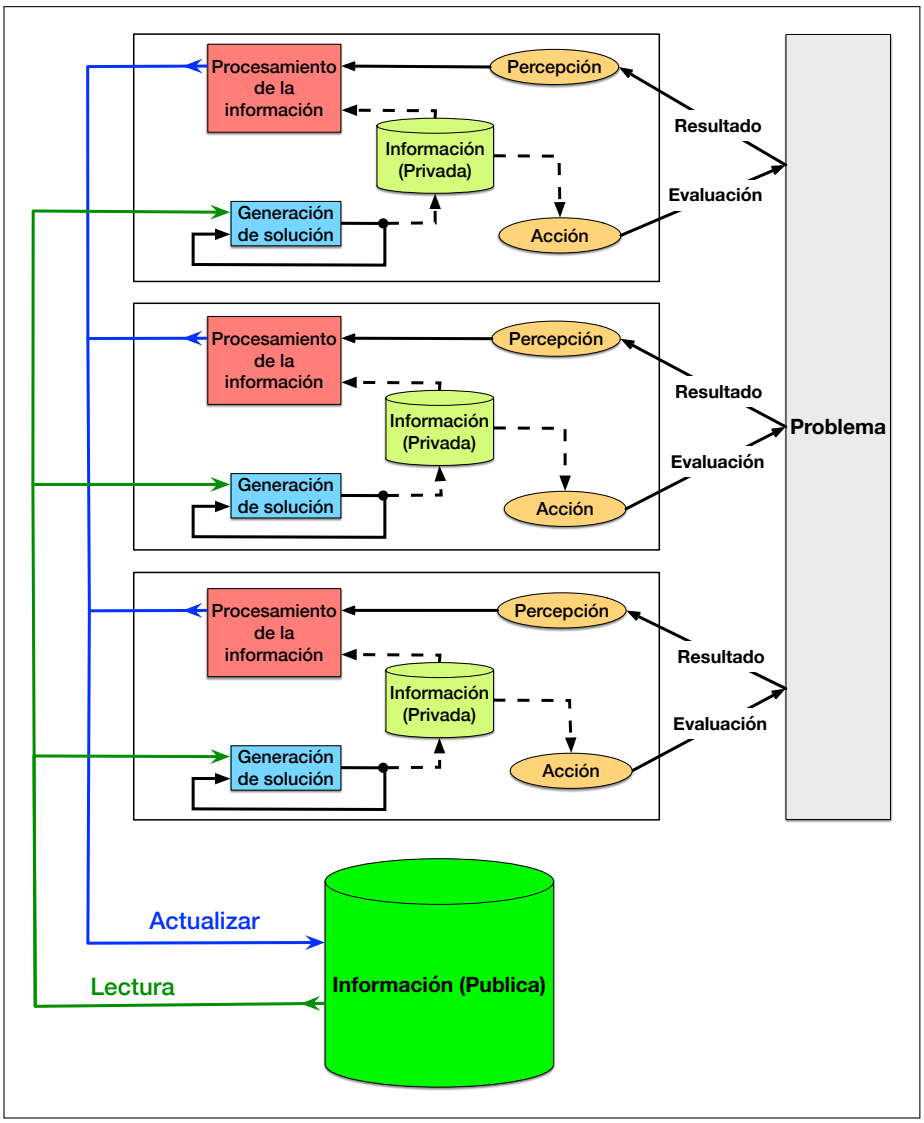


Figura 2.2: Coordinación de agentes mediante información compartida. La generación de soluciones es un bucle que representa el proceso constructivo.

Para explicar el funcionamiento básico de los algoritmos de hormigas, Dorigo utiliza el denominado algoritmo base o simple: S-ACO [DS04].

Simple Ant Colony Optimization (S-ACO)

La metodología ACO se aplica para resolver problemas de optimización combinatorial [DS04], es decir se persigue optimizar la secuencia de elementos que constituyen las soluciones al problema.

Para representar el problema se utiliza un grafo estático, denominado *construction graph*: $G_L = (N, A)$ donde N es el conjunto de vértices del grafo y A es el

conjunto de aristas que conectan los nodos. Cada nodo del grafo $n_i \in N$ constituye un elemento de la solución. Por tanto, partiendo de un cierto nodo inicial n_0, la secuencia de nodos que se recorran y su orden constituyen la secuencia que conforma una solución al problema.

A su vez se definen un conjunto de agentes que conformarán el sistema, denominados hormigas. Cada agente dispone de una estructura de memoria básica, privada, donde almacenan la ruta que han seguido para recorrer el grafo, así como el coste de la misma.

En relación con cada hormiga, en S-ACO definimos dos procedimientos fundamentales:

Construcción de una solución: cada hormiga tiene que construir una solución, una secuencia, debe por tanto recorrer un determinado número de nodos.

Una hormiga qué se encuentre en un determinado nodo i escoge, de manera probabilística, qué nodo visitar a continuación. Esta decisión está basada en información y por ende se dice que ACO es una meta-heurística constructiva: cada elemento que conforma una solución se escoge de manera independiente, es decir, las soluciones se construyen *paso a paso*.

Como el problema lo representamos con un grafo, para determinar qué nodo escoger a continuación podemos asignar un peso a las aristas del grafo. Utilizando estos pesos es posible definir una distribución de probabilidad que será usada por cada hormiga para determinar el nodo que visite a continuación.

Los pesos asociados con una arista se definen mediante dos componentes:

Heurística (η): consiste en un componente *estático*, que se utiliza para incluir información *a priori* acerca del problema y servir de guía para la búsqueda.

Feromona (τ): consiste en un componente *dinámico* que varía a lo largo de la búsqueda. Típicamente, la feromona constituye una tabla (tabla de feromona) que representa la matriz de adyacencia del grafo. Para cada arista del grafo, la tabla contiene un peso que se utiliza para decidir qué nodo se visita a continuación.

Una hormiga situada en el nodo i selecciona qué nodo visitar a continuación utilizando una elección al azar proporcional a los pesos asociados con cada arista. La probabilidad (p_{ij}) de escoger un nodo $j \in N_i$, donde N_i es el vecindario del nodo i, que viene dada por la siguiente ecuación:

$$p_{ij} = \frac{[\tau_{ij}]^{\alpha}[\eta_{ij}]^{\beta}}{\sum_{l \in N_i}[\tau_{il}]^{\alpha}[\eta_{il}]^{\beta}} \tag{2.1}$$

Se puede ver que la ecuación es simplemente una normalización del producto de los dos componentes: heurística y feromona. Cada componente lleva asociado un parámetro: α para la feromona y β para la heurística, para determinar el peso de cada componente en la construcción de la distribución de probabilidad.

Como la heurística es un componente estático, para variar el conjunto de soluciones que se generan, las hormigas deben cambiar los valores asociados a la feromona. Lo que da lugar al segundo procedimiento fundamental en ACO.

Actualización de feromona: este procedimiento se compone de dos pasos: decremento e incremento. El primero se denomina *evaporación de feromona*, el segundo es la actualización propiamente dicha.

Evaporación de feromona: este procedimiento consiste en decrementar *todos* los pesos asociados a alguna arista mediante un valor constante ρ ("pheromone evaporation rate").

$$\tau_{ij} \leftarrow (1-\rho)\tau_{i,j} \quad \forall (i,j) \in A \tag{2.2}$$

Actualización: cuando una hormiga construye una solución, ésta se evalúa mediante la función de coste del problema *J*. De acuerdo con el coste obtenido se obtiene un valor para el incremento de feromona: $\Delta\tau$.

Utilizando este valor de incremento, una hormiga k incrementa la feromona asociada a aquellas aristas que hayan utilizado en su recorrido de la siguiente manera:

$$\tau_{ij} \leftarrow \tau_{i,j} + \Delta\tau^k \quad \forall (i,j) \in T^k \tag{2.3}$$

donde T^k representa el recorrido del grafo realizado por la hormiga k.

Para comprender mejor el funcionamiento de ACO, vamos a presentar un pequeño ejemplo: disponemos de 10 hormigas que eligen siguiendo una distribución de feromona inicialmente plana, representada en la primera iteración de la figura 2.3. Utilizando esta distribución, las hormigas eligen una arista, marcadas con letras desde la *a* a la *j*. A continuación se procede a la actualización de la feromona, donde cada hormiga incrementa la feromona de aquella arista que ha elegido.

Como se trata de un ejemplo, no usaremos ni función de coste, ni heurística. Cada hormiga incrementa el valor de la feromona de manera constante: $\Delta\tau = 0{,}5$ y $\rho = 0{,}1$.

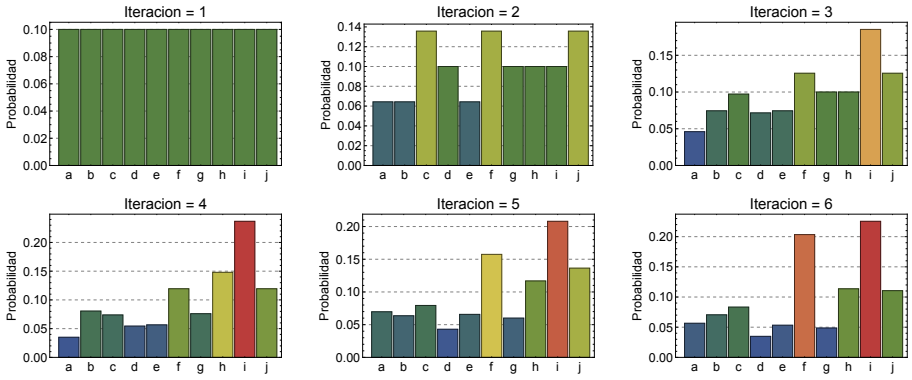


Figura 2.3: Evolución de una distribución de feromona de acuerdo con las ecuaciones de evaporación y actualización. Los valores de feromona se representan ya normalizados (probabilidad).

Como se observa en la figura 2.3, en la iteración 2 existen aristas que no han sido elegidas (a, b, e) y por tanto su valor ha decrecido debido a la evaporación. Otro grupo de aristas (d, g, h, i) ha sido escogido por pocas hormigas, lo que le ha permitido conservar su valor, pero no se ha convertido en la elección dominante.

Por último, existen dos aristas que han sido elegidas con mayor frecuencia (c, f, j), incrementando su valor.

En la iteración 3, las hormigas escogen con la distribución resultante de la iteración 2, y así sucesivamente. Esto permite que cuando alguna de las opciones se desmarca del resto, acumula suficiente feromona, tiende a ser elegida predominantemente en iteraciones sucesivas. Se forma un bucle de realimentación positiva: la decisión más frecuente tiene más probabilidades de volver a ser elegida, permitiendo al algoritmo converger hacia una distribución con unas pocas opciones destacadas sobre el resto de la distribución.

La metodología ACO se ha aplicado a muchos tipos de problemas, incluso optimización real. Entre los problemas donde se ha mostrado más competitiva destacan la búsqueda de camino en grafos [DMC96, DG97, Stu97], planificación de tareas [Blu02, Stü98], y enrutamiento en redes de comunicaciones [DCD97]. Una colección más detallada de las aplicaciones donde ACO ha tenido mejor rendimiento puede encontrase en [DS04].

En el contexto de este estudio no nos interesan tanto los detalles de la metodología de ACO como su filosofía de diseño, en la siguiente sección veremos cómo trasladar algunas ideas clave de Dorigo a otro contexto: la búsqueda en espacios de estados.

2.2. Búsquedas en espacios de estados

Cuando nació la Inteligencia Artificial, ésta perseguía un esquema genérico de solución de problemas, uno de los primeros enfoques que se adoptó fue el que se denomina: "solving problems by searching" [NS+72, RN95]. La metodología parte de la noción de espacio de estados.

En teoría de la computación, el espacio de estados hace referencia a un modelo computacional: un modelo matemático de un supuesto computador. El modelo más simple que se puede encontrar se denomina *máquina de estados finita* o *autómata finito* [Hop07, Sip06].

Partiendo de los términos que conforman la definición de un autómata, se extiende esta definición para obtener lo que se denomina *un problema bien definido*[1] en un espacio de estados:

1. **Espacio de estados**

   $Q_0 \in Q$ el conjunto posible de estados iniciales.

   U el conjunto de acciones (operadores) posibles.

   $\delta : Q \times U \to Q$ la función de transición.

   El **espacio de estados** queda definido como aquellos estados alcanzables desde un estado inicial mediante alguna secuencia de operadores. A su vez se define el término **ruta** (path) como la secuencia de operadores que permiten transitar de un estado a otro.

2. **Criterio de aceptación o parada**, constituye un procedimiento que se aplica para determinar si el proceso de búsqueda se considera finalizado o no. Es la misma idea de los estados de aceptación de un autómata, pero generalizada en forma de procedimiento.

[1] El término *bien definido* fue introducido por primera vez por J. McCarthy [McC56]

3. **Función de coste (*J*)**, es una función que asigna un coste a una ruta determinada.

4. **Definición de solución**, es el criterio que determina qué se considera solución al problema. Por ejemplo, en algunos problemas la solución puede ser el estado final alcanzado, pero también puede ser que estemos interesados en la secuencia de operadores aplicada, etc...

La idea fundamental de esta aproximación clásica es muy sencilla. Al definir el problema como un modelo computacional, la solución que buscamos está contenida en el espacio de estados de dicho modelo. Si conocemos el espacio de estados, conocemos la solución. Por tanto el problema reside *en la construcción del espacio de estados*. Es decir, el espacio se explora construyéndolo. El método de búsqueda no es más que una política, una estrategia, que define cómo ha de construirse dicho espacio.

2.3. Esquema de trabajo

El esquema básico del algoritmo que estamos diseñando se compondrá de dos elementos fundamentales, por un lado tomamos la coordinación de un sistema multi-agente mediante la idea de "Stigmergy" adoptada por Dorigo en la metodología ACO. Pero los agentes, las hormigas, que componen dicho sistema adoptan la aproximación clásica de buscar en un espacio de estados.

El esquema básico del sistema multi-agente que vamos a diseñar se muestra en la figura 2.4. Para comprender mejor como funciona el sistema descrito, vamos a detallar las diferencias comparativamente con la metodología de ACO.

- Los agentes, al igual que en ACO, son procesos iterativos de búsqueda. En cada paso de búsqueda el agente avanza en la obtención de una solución.

  La diferencia es que en ACO, los agentes recorren un nodo, añadiendo un componente a la solución. En nuestro caso los agentes recorren un espacio de estados, de tal modo que en cada paso de búsqueda el agente realiza lo siguiente:

  i) Escoge una acción u en función de su estado actual q.

  ii) En función de la acción escogida cambia de estado: $q^+ = \delta(q, u)$

- En ACO, los agentes almacenaban (memoria privada) la ruta seguida al recorrer el grafo: $[n_0, n_1, \ldots, n_i]$.

  En nuestro caso los agentes almacenan en su memoria privada asociaciones estado-acción que representan las decisiones tomadas:

$$\left| \begin{array}{cccc} q_0 & q_1 & \ldots & q_i \\ u_0 & u_1 & \ldots & u_i \end{array} \right|$$

- En ACO la información pública se define mediante la combinación de *dos componentes* la tabla de feromona y el componente heurístico. Estos dos componentes definen una distribución de pesos acerca de qué nodo visitar a continuación.

En nuestro caso, al trabajar en un espacio de estados, la información debe representar una distribución de pesos acerca de qué acción realizar en función del estado actual. Pero como se ha dicho antes, el algoritmo desconoce el espacio de estados, tiene que construirlo. Por tanto, como se desconoce qué estados constituyen el espacio hasta que se visitan, la información pública la dividimos en *dos distribuciones independientes*:

Información previa o Heurística (η): consiste en una distribución de pesos asociados a las acciones disponibles según el estado actual. La distribución es *estática*.

Si disponemos de conocimiento *a priori* acerca del problema, lo podemos utilizar para definir esta distribución. En caso contrario siempre podemos recurrir a una distribución uniforme (equiprobable) para determinar la elección de la acción.

Información adquirida o Feromona (τ): consiste en el conocimiento que se obtiene acerca del espacio de estados proveniente de la búsqueda de cada agente.

Se trata de una tabla de asociación, donde para cada estado se vincula una lista de pares $[acción, peso]$ que indican la probabilidad de realizar una determinada acción en un determinado estado que haya sido visitado con anterioridad.

En nuestro caso, un agente que escoja una acción nunca combina ambas distribuciones, sino que o bien utiliza la heurística para tomar una decisión, o utiliza la feromona. En caso de que no haya conocimiento adquirido para su estado actual en la tabla de feromona, irremediablemente se recurre a la información heurística.

- Finalmente, en ACO un agente actualizaba la feromona en función de los nodos visitados y la calidad de la solución obtenida. En nuestro caso el procedimiento es similar, pero se incorpora para cada estado visitado la acción realizada y un peso. El peso es función de la calidad de la solución obtenida por el agente.

Como se puede apreciar mantenemos una coordinación basada en la información como en ACO, lo que cambia es el carácter de dicha información. En nuestro caso esa información hace referencia a un espacio de estados y no a un grafo. Además, la información pública no la definimos como la combinación de dos componentes. En nuestro caso la información pública está formada por dos distribuciones independientes: una estática (heurística) y una dinámica (feromona).

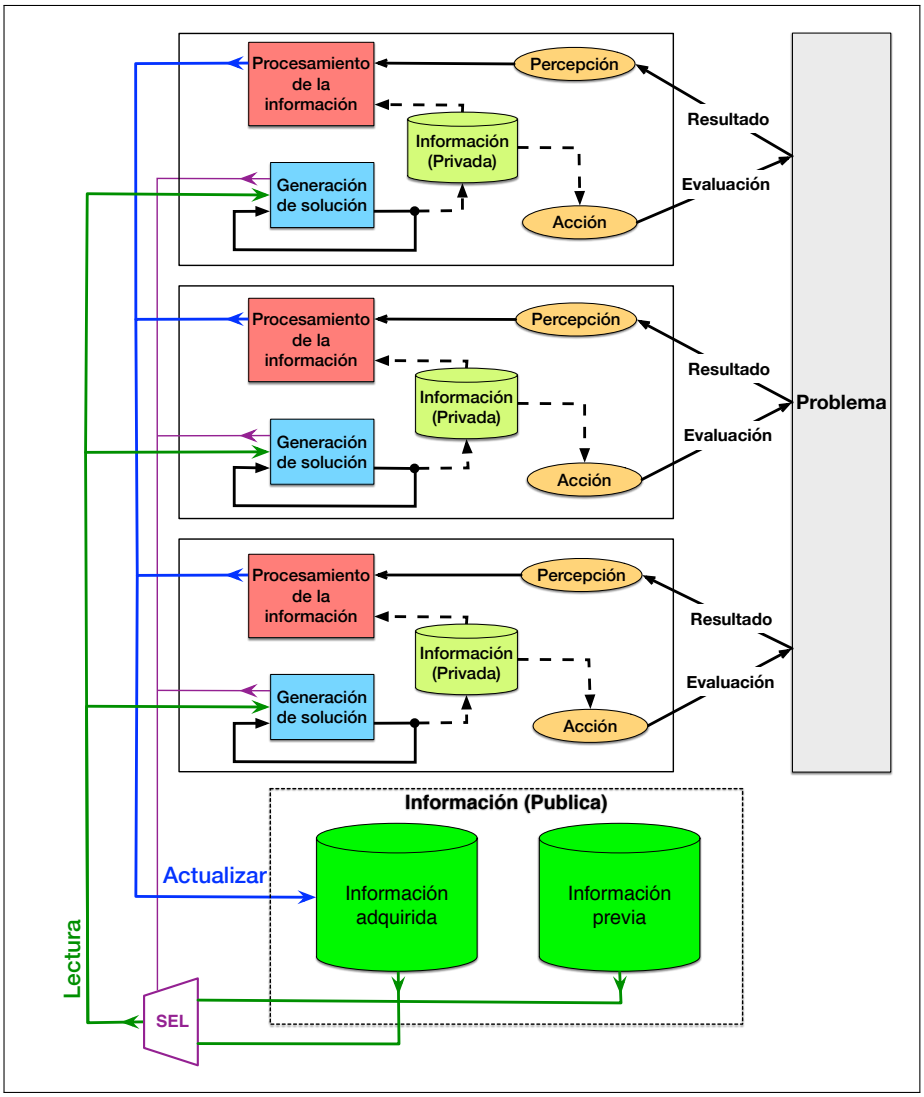


Figura 2.4: Esquema básico de diseño del sistema multi-agente.

2.4. Recapitulación

En estos primeros estadios del diseño hemos tomado dos decisiones importantes: utilizar la filosofía de ACO y una representación de los problemas en espacio de estados. Aunque las razones de estas decisiones se comprenderán mejor cuando avancemos en el diseño, podemos avanzar alguna consideración del porqué de estas decisiones.

Una de la razones de utilizar como base ACO frente a otras meta-heurísticas ya fue introducida previamente: se trata de un sistema multi-agente, que es una técnica típica para simular/estudiar comportamientos auto-organizados. Por tanto, facilitamos el "traslado" desde los modelos biológicos que vamos a ver a continuación, al esquema de algoritmo que estamos diseñando.

La segunda razón tiene que ver con que se trate de una meta-heurística constructiva. En las otras clases de meta-heurísticas, búsqueda y evolutivas, el conocimiento del problema se representa mediante una colección de soluciones del problema, una nueva solución se obtiene por transformaciones de anteriores. No existe una separación definida entre el conocimiento y la generación de soluciones. Esto puede plantear problemas añadidos a la búsqueda.

En nuestro caso preferimos un esquema donde el conocimiento y la solución sean elementos separados. Obviamente existe una relación, pues el conocimiento determina el tipo de solución que un algoritmo genera. Pero al tratarlos por separado, no existe un vínculo tan fuerte, ya que permitimos que exista conocimiento que no se use. Es decir, un algoritmo puede disponer de un conocimiento amplio acerca de un problema y en cambio producir un patrón claro de soluciones, basta con que únicamente utilice una parte del conocimiento del que dispone para generar soluciones.

Al separar conocimiento y solución en un esquema constructivo, obtenemos una flexibilidad que es más difícil de obtener si optamos por un esquema donde la solución represente el conocimiento.

Éstas son las razones fundamentales por las que escogimos ACO como esquema de partida. Respecto al uso una aproximación clásica de espacio de estados, las razones son igualmente dos. La primera es que un grafo puede ser una representación complicada según qué tipos de problemas. Además exige su construcción previamente a la aplicación del algoritmo, si el grafo tiene un tamaño considerable comenzamos a tener problemas de rendimiento (tiempo de cómputo) y convergencia. Recordemos que la feromona inicialmente es una distribución plana que el algoritmo hace converger hacia unas pocas opciones, si dicha distribución tiene miles de opciones (cada nodo tiene miles de vecinos), la convergencia puede resultar bastante complicada.

Al utilizar un espacio de estados abrimos la metodología a otra clase de problemas sin perder los anteriores, ya que un grafo es una manera de definir, representar un espacio de estados. En problemas donde el grafo no sea una representación adecuada, es posible que un espacio de estados si lo sea.

Además el espacio de estados no necesita construirse previamente, sino que se va *descubriendo*. Esto implica que ya no partimos de una distribución plana que hacer converger, sino de una distribución "vacía" que vamos a ir "llenando". Por tanto, podemos evitar más fácilmente el problema de rendimiento en problemas complejos (tiempo de cómputo), e igualmente el problema de dificultad de convergencia.

La segunda razón es que la aproximación en espacio de estados nos permite

definir la información pública como dos distribuciones independientes. Al tratarse de agentes, permite convertir el problema del balance de la búsqueda (exploración/explotación) en términos de actividad: dependiendo de la frecuencia de uso de cada tipo de información, heurística o feromona, la búsqueda global puede resultar más o menos dispersa.

Como veremos en el capítulo siguiente, las dinámicas auto-organizativas biológicas están dirigidas por la actividad que un individuo del sistema realice en su entorno. Por tanto, si conseguimos expresar el problema del balance de la búsqueda en términos de actividad, podemos expresar fácilmente las propiedades auto-organizativas de los sistemas estudiados en el contexto de un algoritmo.

Capítulo 3

Decisiones colectivas en los insectos

En este capítulo vamos a desarrollar un prototipo de algoritmo utilizando el esquema de sistema multi-agente visto anteriormente. Este prototipo estará inspirado en la inteligencia colectiva que muestran las colonias de insectos. Cada insecto de manera aislada se comporta de una manera mecánica, es decir, no se puede calificar como inteligente de la misma manera que calificamos un perro o un delfín. Pero cuando los insectos se agrupan en colonias, la colonia es capaz de resolver problemas complejos que requieren de inteligencia.

Los problemas a que se enfrenta una colonia de insectos son variados, en nuestro caso nos vamos a centrar en la toma de decisiones. Vamos a estudiar dos tipos de colonias distintas: las hormigas y las abejas, centrándonos en los elementos más atractivos de cara a diseñar un algoritmo.

3.1. Colonias de hormigas

La *formación de senderos de feromona* que llevan a cabo las colonias de hormigas inspiró a M. Dorigo a desarrollar la metodología ACO. Nosotros nos vamos a centrar igualmente en esta característica, pero poniéndola en relación con las colonias de abejas. Por esta razón hemos separado la inspiración biológica de la presentación de ACO.

La feromona [HW90, HW09] es una sustancia química segregada por las hormigas para comunicarse entre ellas. Aunque se suele hablar de comunicación, realmente es más un proceso de organización que una comunicación estándar, porque la feromona únicamente puede variar en concentración. Al variar de concentración, la feromona permite dejar un huella y por tanto establecer rutas:

- Cuando una hormiga se mueve, segrega feromona quedando depositada en el terreno.
- Cuando una hormiga debe decidir en que dirección moverse, tiende a escoger aquellas direcciones donde la concentración de feromona es mayor.

Como se puede apreciar, las hormigas se mueven siguiendo huellas químicas. Este tipo de comportamiento se suele denominar comunicación indirecta basada en el

entorno: un individuo es capaz de percibir en el entorno la acción que previamente ha realizado otro individuo. Se establece un vínculo entre los individuos, pero sin necesidad de transmitir información explícita entre ambos. Por ejemplo, una hormiga no sabe donde fue la hormiga anterior, ni sus intenciones, etc.... Únicamente es capaz de percibir que en el mismo lugar donde ella se encuentra, otras hormigas avanzaron en una dirección determinada.

Una manera de ilustrar la capacidad de la toma de decisiones colectivas de las hormigas es con una sencilla simulación. Supongamos que la hormiga avanza desde el hormiguero hacia delante buscando una fuente de comida, este movimiento es al azar guiado por la concentración de feromona. Cuando la hormiga encuentra comida, vuelve al hormiguero depositando una cierta cantidad de feromona. La simulación se muestra en la figura 3.1, en las sub-figuras superiores se muestra la fuente de comida (cuadrado rojo), el hormiguero (azul), las hormigas que avanzan (negro) y las que vuelven (rojo). En las sub-figuras inferiores se muestra como evoluciona la concentración de feromona.

Como se puede ver las hormigas salen del hormiguero de manera continúa, al principio como no existe ninguna marca de feromona, este movimiento es al azar; por lo que algunas hormigas dan con la comida y otras simplemente se pierden. A medida que empiezan a encontrar la fuente con más frecuencia, empieza a incrementarse la concentración de feromona significativamente, permitiendo un movimiento más efectivo. Finalmente la ruta se establece perfectamente como se puede ver en la línea –casi– recta que une el hormiguero y la comida.

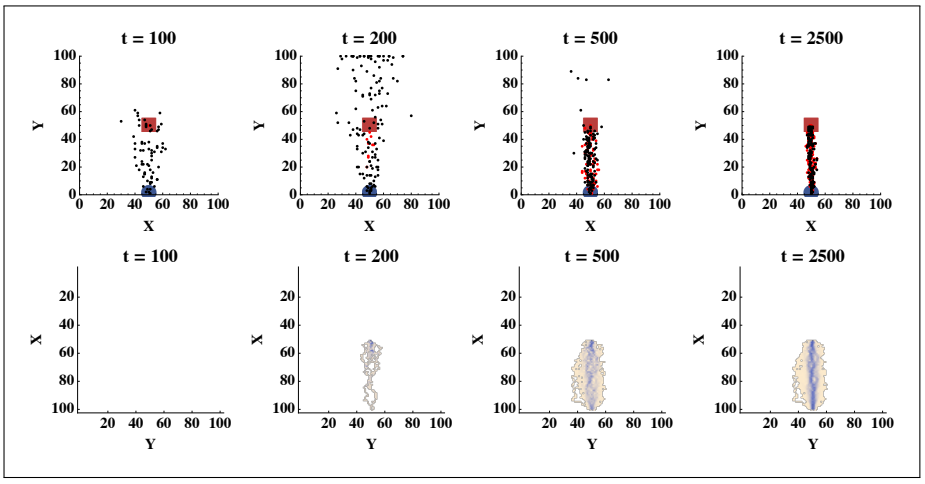


Figura 3.1: Formación de senderos de feromona.

La clave del comportamiento de las hormigas reside en el refuerzo positivo que se establece con la feromona: se escoge preferentemente aquellas direcciones con mayor concentración. Como al avanzar depositan feromona, al elegir una dirección determinada, se incrementa la concentración de feromona de dicha dirección. De tal modo que cuanto más veces se elija cierta dirección, mayor concentración de feromona será depositada y más probable resulta la elección de dicha dirección en el futuro. Este mecanismo de refuerzo permite a las hormigas escoger de manera colectiva una ruta entre la fuente de comida y su hormiguero. Es colectiva porque dicha ruta no ha sido establecida por una hormiga, sino por muchas.

El experimento del doble puente

Este experimento fue desarrollado por Goss, Aron, Deneubourg y Pasteels [GADP89]. El experimento consistía en utilizar un doble puente para conectar una colonia de hormigas de la especie Argentina I. humilis y una fuente de comida. Realizaron los experimentos variando el ratio $r = \frac{l_1}{l_2}$ que es la longitud entre los dos ramales del puente. l_1 es la longitud del más largo y l_2 la del más corto. Las condiciones del experimento pueden verse en la figura 3.2.

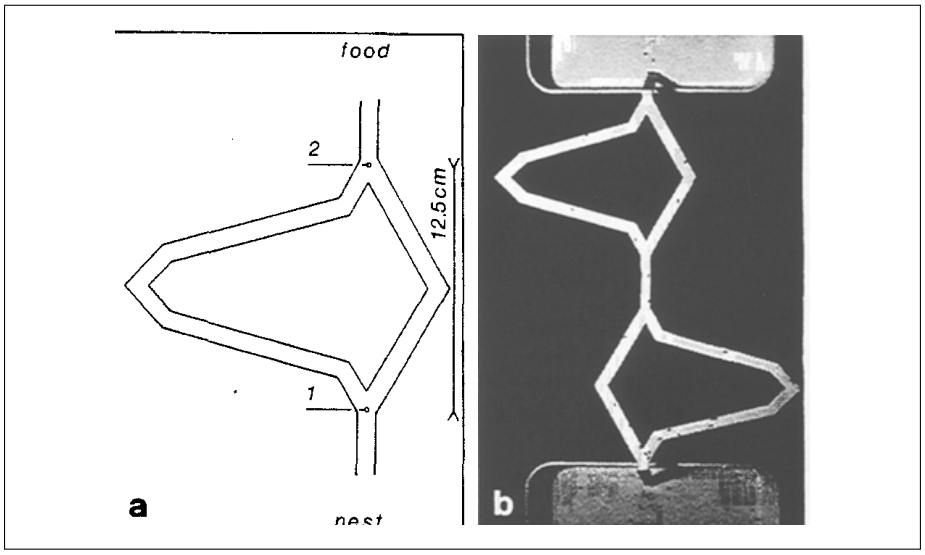


Figura 3.2: Experimento del doble puente (Imagen tomada de [GADP89])

En un primer experimento, los investigadores presentaron a las hormigas los dos ramales desde el inicio. Transcurrido cierto tiempo las hormigas alcanzaban la fuente de comida utilizando los ramales más cortos. La explicación es sencilla: consideremos que la segregación de feromona por parte de una hormiga es constante. Por tanto en un cierto intervalo de tiempo Δt el camino más corto presentará una concentración mayor, simplemente porque al ser más corto se ha podido recorrer un mayor número de veces. Al presentar una mayor concentración, cuando "nuevas" hormigas parten del hormiguero, escogerán preferentemente el ramal corto, pues su concentración de feromona es superior. Se puede intuir que transcurrido más tiempo, se establece una ruta entre el hormiguero y la comida mediante los ramales más cortos.

En un segundo experimento, los ramales largos estaban disponibles desde el inicio y los cortos fueron introducidos después. En este caso la ruta se estableció utilizando los ramales largos. Al ser la única opción disponible desde el inicio, acumularon una concentración de feromona tal que hizo imposible la elección del ramal más corto cuando éste fue introducido.

Esta serie de experimentos muestra la capacidad de toma de decisiones de las hormigas: entre todas las rutas posibles para alcanzar una fuente de comida, una colonia tiende a seleccionar la que presenta una menor longitud. Ahora bien, el experimento también revela que dicha decisión es fija: aunque las condiciones

varíen, una decisión tomada es inamovible.

3.2. Colonias de abejas

Al igual que en el caso de las colonias de hormigas, nos centraremos en la toma de decisiones en las colonias de abejas. El mecanismo es más complejo, ya que no existe feromona, los individuos se comunican directamente mediante una danza.

El investigador Karl R. von Frisch [VF67] fue quien estudió y descubrió el mecanismo de comunicación en las abejas a través de lo que él denominó "danza". La danza consiste en una serie de movimientos y desplazamientos realizados por una abeja, los cuales son observados e interpretados por otras abejas. Mediante estos movimientos, una abeja es capaz de transmitir la dirección y distancia de una fuente de polen y néctar.

Dejando la danza a un lado, nos centraremos en cómo una colonia de abejas decide qué fuente explotar cuando existen varias disponibles.

Toma de decisiones colectivas

El modelo de decisiones colectivas en las colonias de abejas [SCS91, CDF+03] se puede representar gráficamente de una manera sencilla.

La figura 3.3 representa dicho modelo para el caso de que existan dos fuentes de néctar. Existen diferentes elementos básicos: las fuentes de néctar, la zona de danza (indicada con un circulo gris) y la colmena. A su vez también existen diferentes estados para una abeja:

Comunicación: son aquellas abejas que tras explotar una fuente de néctar ingresan en la zona de danza para transmitir la información acerca de esta fuente.

Observación: son aquellas abejas que se encuentran en la zona de danza recibiendo información (observando el baile) acerca de las fuentes de néctar en explotación.

Una observadora elige al azar entre la totalidad de fuentes que le están siendo comunicadas. Es decir, únicamente se fija en un baile.

Recolectoras: son aquellas abejas que están explotando una fuente de néctar. La explotación de una fuente puede conducir a un proceso de danza (comunicación) o a un abandono de la recolección (inactividad).

Inactiva: son aquellas abejas que se encuentra en el interior de la colmena. Dichas abejas pueden activarse para ingresar en la zona de danza y convertirse en observadoras. O bien pueden convertirse en abejas "scout" (exploradoras), las cuales parten en busca de una posible fuente de néctar a explotar.

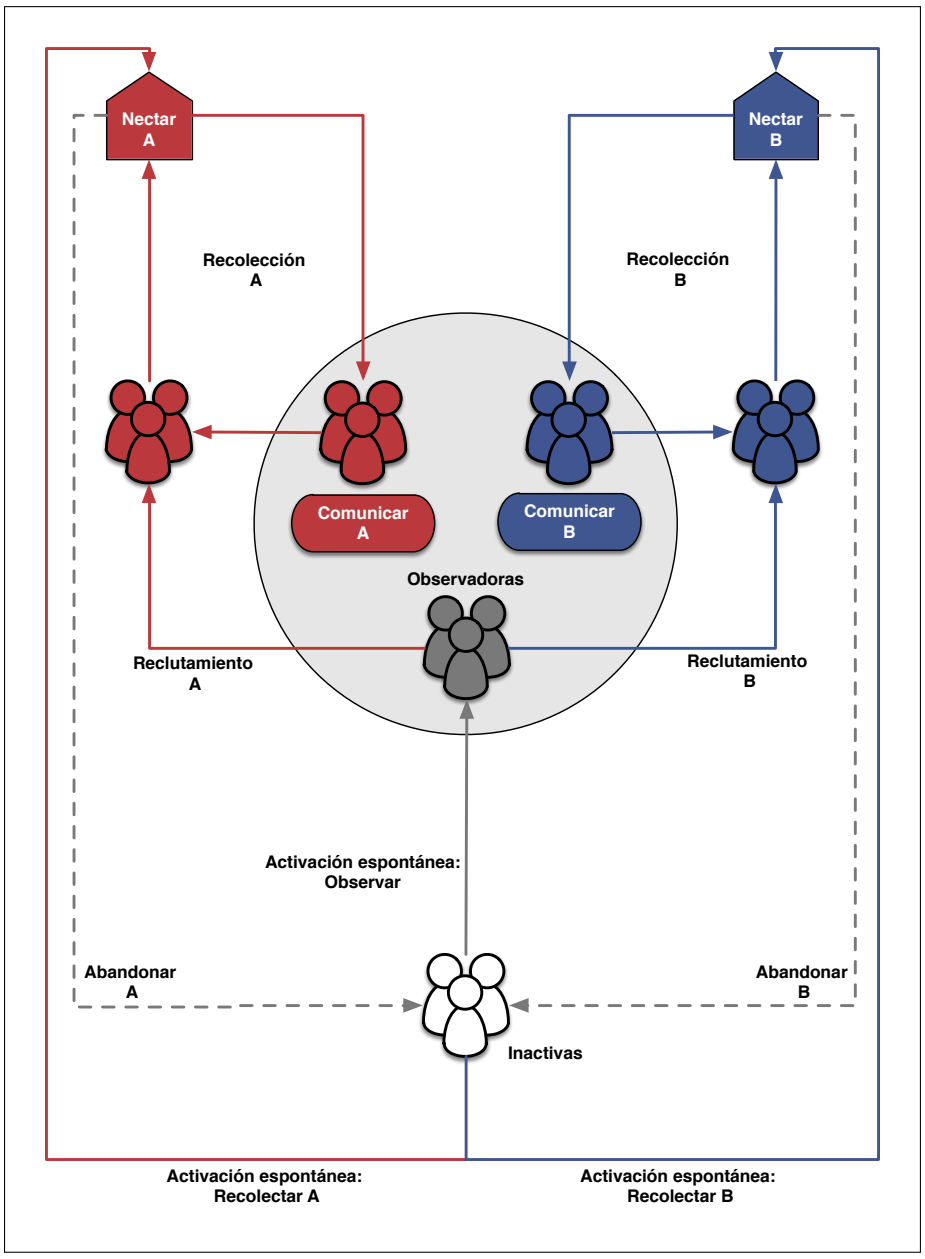


Figura 3.3: Modelo de toma de decisiones colectivas en una colonia de abejas.

Cuando encuentra una fuente de néctar, dependiendo de la concentración de glucosa de la fuente una abeja puede tomar diferentes decisiones :

a) abandonar la recolección.

b) continuar con la recolección sin comunicar su localización.

c) comunicar la localización de la fuente de néctar. Una vez que termina de comunicar (termina la danza), la abeja continua con la recolección.

Para facilitar la explicación, vamos a reducir las decisiones que toma una abeja a únicamente dos tipos: abandonar y comunicar. Teniendo esto en cuenta, la actividad de recolección de néctar la podemos resumir de la siguiente manera:

- El proceso comienza con las denominadas abejas "scout" o exploradoras. Las cuales, de manera espontánea, parten a explorar los alrededores en busca de una fuente de néctar.
- Cuando una abeja encuentra una fuente de néctar comienza a explotarla. En base a la concentración de glucosa, la abeja puede:
  a) Abandonar la fuente de néctar, volviendo al interior de la colmena (inactiva).
  b) Comunicar la localización de la fuente de néctar. En este caso produce un proceso de *reclutamiento*: las abejas observadores pueden adquirir el conocimiento acerca de la localización del néctar, explotando a su vez la misma fuente.
- Las abejas inactivas, periódicamente, ingresan en la zona de baile pasando a ser observadoras. Esto permite la transición entre abejas inactivas y observadoras, aumentando el número de abejas dedicadas a la recolección de néctar.

La decisión colectiva se realiza por medio de un mecanismo de refuerzo positivo:

i) Una abeja comunica la posición de una fuente de néctar X.

ii) Otra abeja observadora recoge esa información y comienza a explotar la fuente X.

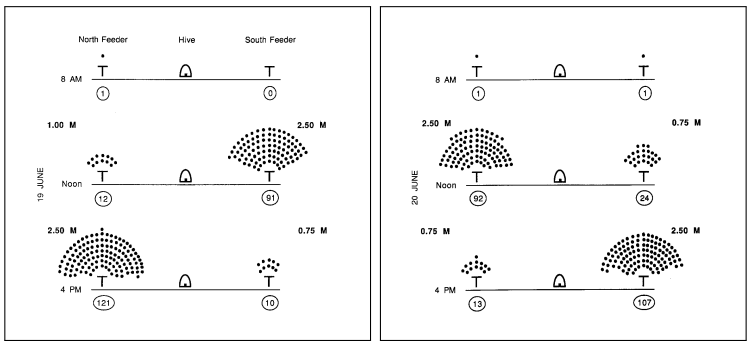
iii) Existen dos abejas explotando y comunicando la localización de la fuente X. Lo cual incrementa las probabilidades de que otras abejas se unan a la explotación de la fuente X.

Como se puede apreciar el refuerzo positivo consiste en que cuantas más abejas exploten una fuente, más abejas comunicarán la localización de dicha fuente y más abejas se unirán a la explotación de la misma fuente.

Experimento real

En el estudio de campo [SCS91] que investigaba la toma de decisiones colectivas, los investigadores plantearon el siguiente experimento: dos fuentes de néctar (sur y norte), con diferentes concentraciones de glucosa. Una vez que las abejas se habían decidió por una de ellas, intercambiaban las concentraciones. El esquema del experimento puede verse en la figura 3.4.

En un principio la abejas encuentran ambas fuentes con igual probabilidad. Como existen diferentes concentraciones de glucosa, para la fuente de concentración más alta existe una mayor probabilidad de comunicar su posición que de abandonar la recolección. Para la fuente más baja ocurre lo contrario. Poco a poco, las abejas que descubren la fuente de mayor concentración, reclutan más abejas para su recolección. Finalmente la colonia en su totalidad está explotando la mejor de las fuentes disponibles.



(a) Alta concentración de glucosa inicial en la fuente sur.

(b) Alta concentración de glucosa inicial en la fuente norte.

Figura 3.4: Experimento con diferentes fuentes de néctar (Imagen tomada de [SCS91]).

Lo más interesante de este experimento es que cuando los investigadores modificaron las concentraciones, las abejas cambiaron la decisión sin apenas dificultad. Esta capacidad de las abejas contrasta con las colonias de hormigas, las cuales eran incapaces de cambiar una decisión ya tomada.

La capacidad de cambio de las abejas se debe a que los individuos funcionan de manera asíncrona: mientras unas abejas explotan una fuente de néctar, otras están bailando. Como la comunicación, la transmisión de información, es directa (individuo a individuo), la información presenta un carácter temporal: únicamente estará disponible mientras la abeja baila. Este componente temporal incrementa la flexibilidad del proceso de toma de decisiones. En una colonia de hormigas, la información se deposita en el medio y permanece accesible ininterrumpidamente.

3.3. Hormigas + Abejas

Hemos presentado dos modelos de toma de decisiones colectivas, las colonias de hormigas y las colonias de abejas. El objetivo es obtener un diseño de algoritmo básico que permita combinar las virtudes de ambos modelos, para lo cual destacaremos sus aspectos fundamentales.

Hormigas: información local

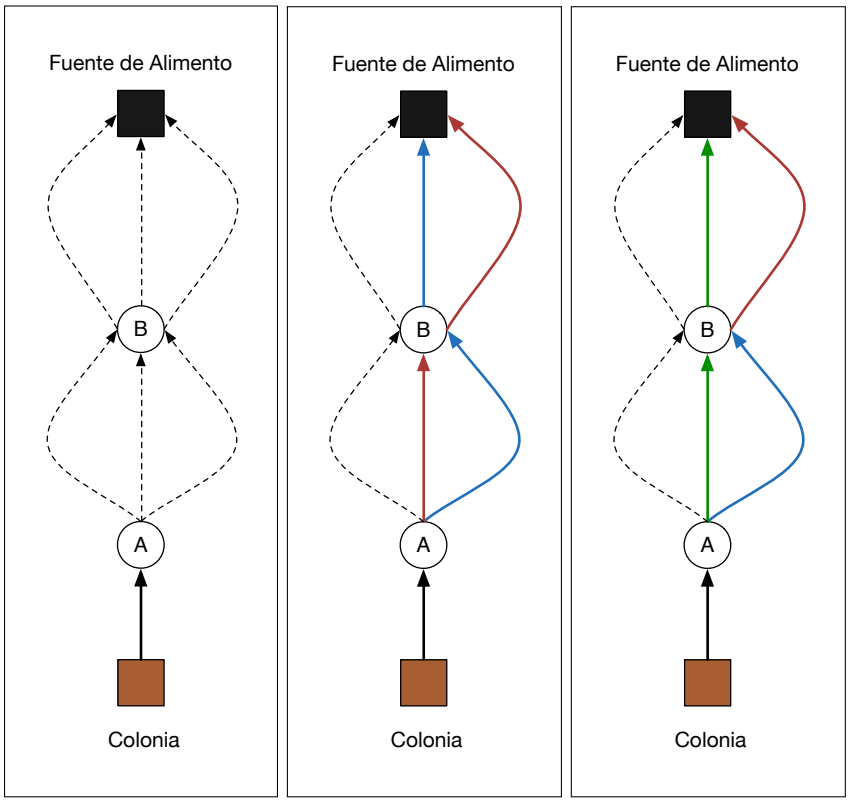
Lo más interesante de la toma de decisiones basada en una marca de feromona es la asociación de la información a cada decisión individual que debe realizar una hormiga para llevar a cabo un cierto recorrido. Denominaremos a esta característica "información local".

La mejor manera de entender las bondades de la información local es mediante un ejemplo sencillo. En la figura 3.5 se muestra un escenario para alcanzar una fuente de alimento desde una colonia. Existen dos puntos de decisión: A y B, en cada punto existen tres decisiones posibles: avanzar en línea recta o dar un rodeo (izquierda o derecha). Obviamente la mejor solución es avanzar en línea

recta. Veamos cómo podemos alcanzar este conocimiento suponiendo que fuésemos hormigas.

La posibilidad más obvia es que en cada decisión escogiésemos el camino en línea recta, de este modo alcanzaríamos la comida de manera óptima. Pero esta no es la única posibilidad.

Supongamos que dos hormigas han alcanzado la comida como se muestra en la figura 3.5, seleccionado un tramo recto y dando un rodeo. ¿Qué ocurre con la siguiente hormiga? Las hormigas, marcadas como rojo y azul, han dejado un conocimiento en cada punto de decisión. Por tanto, la hormiga que venga posteriormente (verde) puede utilizar el conocimiento de la hormiga roja en el punto *A* y el conocimiento de la hormiga azul en el punto *B*. Es otra forma de conseguir alcanzar la comida de manera óptima.



(a) Situación inicial. (b) Primeros recorridos. (c) Recorridos posteriores.

Figura 3.5: Ejemplo sencillo de la información local en la toma de decisiones colectivas. Las lineas discontínuas indican posibles caminos para alcanzar la fuente de comida, cada color representa el camino que ha seguido una hormiga diferente.

La información local hace referencia a cada decisión de manera independiente, por tanto permite combinar conocimiento de decisiones tomadas por individuos diferentes y obtener un comportamiento nuevo.

En nuestro ejemplo, significa combinar las decisiones de las rutas de las hormigas roja y azul para obtener una ruta verde. Encontrar el camino recto sin recurrir

al conocimiento tiene una probabilidad de $\frac{1}{3} \cdot \frac{1}{3} = \frac{1}{6}$. Al utilizar el conocimiento esta probabilidad aumenta $\frac{1}{2} \cdot \frac{1}{2} = \frac{1}{4}$.

En un algoritmo constructivo el uso de información local, por ejemplo vinculando las acciones a los estados, permite hacer uso de la información generada por un agente en el proceso de construcción de su solución, por otro agente distinto que construye una solución diferente, únicamente es necesario coincidir en el mismo estado, en el mismo punto de decisión.

Abejas: información temporal

Lo más interesante del modelo de las abejas es su capacidad de cambiar la fuente de néctar que están explotando cuando las concentraciones cambian. El proceso de decisión de las abejas converge a una solución, igual que las hormigas convergen a un cierto camino. Pero si las condiciones cambian, las abejas son capaces de alterar su decisión. Por el contrario las hormigas se quedan "atascadas".

La diferencia entre ambos comportamientos se explica fundamentalmente atendiendo a los procesos de comunicación de cada colonia de insectos. En las hormigas la comunicación es a través del medio: una hormiga deja una marca de acuerdo con la decisión tomada. Esto produce que la información quede adscrita al medio: no desaparece. Con lo cual, cuando una hormiga toma una decisión accede a la totalidad de información que exista acerca de esa decisión: a todas las marcas dejadas por el resto de hormigas previamente.

En la comunicación de una colonia de hormigas, la conectividad es "máxima": todas las hormigas tiene comunicación con el resto de la colonia. Esto produce una convergencia rápida hacia un tipo de patrón determinado, por ejemplo una fila de hormigas. Pero además implica que dicho patrón sea estable, porque el ratio de acumulación de feromona en el patrón predominante es tan alto, que impide la acumulación significativa de concentración en patrones de conducta minoritarios.

Por ejemplo, cuando una hormiga 'trata" de salirse de la fila, la influencia del resto de hormigas (las que continúan en la fila) enmascara cualquier rastro simplemente porque son muchas hormigas "haciendo otra cosa".

En las colonias de abejas, la conducta no se adquiere según información local adscrita a cada decisión. Las abejas adquieren toda la información "de golpe" en la zona de baile; una vez adquirida la información la aplican para determinar la localización de la fuente de néctar. La conectividad es "limitada": el proceso de comunicación se establece entre una abeja que baila y otras que observan, una vez finalizada la danza, el proceso finaliza y la información comunicada deja de estar disponible.

Supongamos tres abejas, cada una explotando una fuente de néctar distinta: cada abeja tarda 5 segundos en recolectar el polen y después danza durante 5 segundos para comunicar la posición de la fuente. La abeja 1 empieza en $t = 0$, la abeja 2 en $t = 1$ y la abeja 3 en $t = 2$. En la figura 3.6 se ilustra la evolución en el tiempo de los estados de las abejas. Como se puede observar existen instantes de tiempo donde sólo hay una abeja reclutando, en otros instantes hay dos y en otros están todas. Por tanto, dependiendo de cuando aparezca un observador la información a la que tiene acceso es distinta.

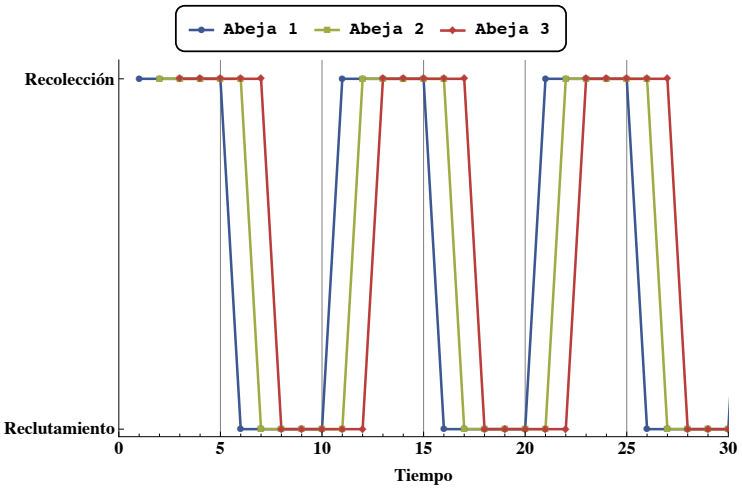


Figura 3.6: Diferentes instantes de reclutamiento de tres abejas distintas.

Si las tres abejas del supuesto indican fuentes de néctar distintas, la colonia puede establecer un patrón de recolección que abarque las tres fuentes. Es decir, la toma de decisiones puede converge a una conjunto de soluciones múltiple, en vez de a una única solución. Igualmente si una abeja encuentra una fuente de néctar desconocida y comunica su decisión a la colonia, esta información no se enmascara por el resto de las abejas que estén asignadas a otra fuente. La información nueva únicamente "compite" con la información que esté siendo comunicada en ese mismo instante de tiempo.

Los patrones minoritarios de conducta no tienen que hacer frente al patrón mayoritario. Al estar distribuido en el tiempo, la superposición se da entre el patrón minoritario y un *fragmento* del mayoritario. Esto facilita la inclusión de nuevas conductas en la dinámica de la colonia. Como la dinámica es un refuerzo positivo, una vez que la nueva conducta se consolida, puede extenderse y convertirse en el nuevo patrón mayoritario de conducta.

Si recordamos, en el capítulo de Introducción, se mencionó la idea de reducir la conectividad entre los agentes de un sistema para aumentar su capacidad de procesar información, debido a que los agentes podían converger localmente en grupos. Esto es justamente lo que ocurre en las colonias de abejas ya que la información presenta una *ventana temporal* que reduce la conectividad entre los distintos miembros de una colmena.

Esta propiedad nos parece realmente interesante para un algoritmo meta-heurístico, ya que al tratarse de procesos iterativos que convergen a un patrón de soluciones, este tipo de dinámica, propia de una colonia de abejas, puede permitir al algoritmo manejar un conjunto mayor de soluciones flexibilizando su convergencia.

Resumen: cada modelo biológico presenta sus ventajas y sus desventajas. Las hormigas son capaces de combinar información entre ellas, pero no existe el componente temporal que otorga esa flexibilidad para cambiar una decisión ya tomada. Por el contrario las abejas pueden cambiar su decisión en cualquier momento, e inclusive decidirse por varias opciones simultáneas. Por contra, no existe locali-

dad en la información: la secuencia de acciones se adquiere en la zona de baile y después se aplica.

La idea es combinar ambos modelos desarrollando un esquema híbrido. La clave está en convertir la tabla de feromona típica de ACO en un "buffer": una estructura de datos reservada para el almacenamiento temporal de información mientras espera ser procesada. De este modo combinamos la localidad de las hormigas con la temporalidad de las abejas.

Para ilustrar esta idea vamos a realizar una simulación sencilla a modo de ejemplo de un sistema multi-agente básico.

3.4. Simulación de un esquema híbrido

La idea del esquema híbrido es construir un sistema multi-agente simple que combina las dos propiedades fundamentales de la toma de decisiones en los modelos biológicos de las hormigas y de las abejas. Para estudiar estas propiedades, vamos a definir un sistema simple a modo de ejemplo.

La simplificación fundamental de este ejemplo consiste en definir un sistema convirtiendo la tabla típica de los algoritmos de hormigas en un especie de "buffer", es decir en un almacenamiento temporal. Para convertir la tabla en un "buffer" eliminamos las probabilidades, definiendo una política de actualización por sobreescritura: cuando la tabla se actualiza la información se remplaza completamente. De este modo, la información tiene carácter temporal, únicamente estará disponible hasta que se realice una nueva actualización.

Descripción

El sistema consiste en un conjunto de agentes coordinados mediante una tabla que almacena información compartida.

El cometido del sistema es simplemente generar secuencias simbólicas, el estado del agente vendría indicado por la posición de la secuencia donde se generaría el símbolo siguiente. Por ejemplo, el agente $A = \{A_0, A_1\}$ estaría en el estado 3. La tabla funciona asociando símbolos con posiciones, al igual que en un problema real asociaría acciones a estados.

Más concretamente:

- El sistema se compone de n agentes.

  Cada agente es capaz de construir una secuencia simbólica de longitud l. La construcción de dicha secuencia se realiza paso a paso, añadiendo un símbolo detrás de otro hasta que se alcanza la longitud l.

- El sistema dispone a su vez de una tabla de feromona que representa la información adquirida. La tabla tiene como clave una posición de la secuencia y como valor asociado un símbolo cualquiera:

Posición	símbolo
1	s_0
2	s_1
3	s_2
⋮	⋮
l	s_l

- Cuando un agente esté construyendo una secuencia, el símbolo que añade será el indicado por la tabla para la posición que tiene que rellenar actualmente. En caso de que no exista información de la tabla, un agente generará el símbolo X_i donde X es la letra asignada al agente e i la posición de la secuencia donde se añade el siguiente símbolo.

- La actualización de la tabla se lleva a cabo por sobre-escritura. Es decir, para cada posición de la secuencia generada por el agente, se escribe –para su correspondiente posición– el símbolo que ocupa dicha posición en la secuencia.

  Por ejemplo, supongamos un agente que genera la secuencia $[A, B]$, dicho agente actualizaría la tabla de la siguiente manera: $1 \rightarrow A, 2 \rightarrow B$.

- El estado inicial del sistema, previamente a que se haya actualizado la información por primera vez, consiste en una tabla vacía. Los agentes comienzan en un punto determinado del proceso de construcción de una secuencia.

El algoritmo que implementa este sistema se ilustra en la figura 3.7. Primeramente se inicializa el sistema, a continuación entramos en el bucle principal que consiste en dos etapas:

a) Actualización de la tabla. En una primera etapa se actualiza la tabla por sobre-escritura para cada agente que haya finalizado la construcción de una secuencia. A su vez, dicho agente se reinicia para volver a construir una secuencia.

b) Una segunda etapa consiste en que cada agente añade un nuevo símbolo a su secuencia utilizando el contenido de la tabla.

```
1:  agentes = Inicialización
2:  tabla = []
3:  while i <N do
4:      for a ∈ agentes do
5:          if finalizado(a) then
6:              tabla = sobre_escribe(a)        ▷ Sobre-escritura de la tabla
7:              a = []
8:          end if
9:      end for
10:     for a ∈ agentes do
11:         a = símbolo(tabla,a)
12:     end for
13:     i = i + 1
14: end while
```

Figura 3.7: Esquema algorítmico utilizado en las simulaciones.

Para facilitar la compresión del esquema se detallan unos breves pasos para el caso de tres agentes: A, B y C que construyen secuencias de tres elementos. En la tabla3.1 se detallan los dos primeros pasos de simulación. Es importante tener

en cuenta que la tabla se actualiza previamente a la construcción de la secuencia por parte de los agentes.

	Agentes			Tabla	
A	A_1	A_2	-	Posición 1	-
B	B_1	-	-	Posición 2	-
C	-	-	-	Posición 3	-

(a) Iteración 0 (estado inicial).

	Agentes			Tabla	
A	A_1	A_2	A_3	Posición 1	-
B	B_1	B_2	-	Posición 2	-
C	C_1	-	-	Posición 3	-

(b) Iteración 1.

	Agentes			Tabla	
A	A_1	-	-	Posición 1	A_1
B	B_1	B_2	A_3	Posición 2	A_2
C	C_1	A_2	-	Posición 3	A_3

(c) Iteración 2.

Tabla 3.1: Simulación de dos iteraciones.

Un sistema en el que todos los agentes comienzan en un mismo estado es un sistema síncrono, los agentes actualizan la información al mismo tiempo (misma iteración), acceden a la misma información, etc... No existe una componente temporal porque todo ocurre siempre con la misma frecuencia.

Un sistema donde todos los agente comienzan en un estado distinto es un sistema asíncrono, en cada iteración un agente diferente actualiza la tabla, cada agente accede a la información en diferentes iteraciones, etc... Existe una componente temporal.

Hay que recordar que estamos estudiando un esquema híbrido, concretamente nos interesa el sistema asíncrono porque permite la aparición de una componente temporal. El esquema síncrono es más típico de algoritmos tipo ACO.

Simulación: componente temporal

Las simulaciones se van a realizar para el caso concreto donde el sistema está compuesto por 4 agentes, los cuales construyen secuencias de 4 símbolos pertenecientes al alfabeto: $\{A, B, C, D\}$. La simulación consiste en evolucionar el sistema durante 50 iteraciones.

El objetivo es analizar la convergencia del modelo según se trate de un esquema síncrono o asíncrono, para lo cual partimos de dos inicializaciones distintas mostradas en la tabla 3.2.

	Asíncrono				Síncrono			
Agente A	A_0	A_1	A_2	A_3	A_1	A_1	A_2	A_3
Agente B	B_0	B_1	B_2	-	B_0	B_1	B_2	B_3
Agente C	C_0	C_1	-	-	C_0	C_1	C_2	C_3
Agente D	D_0	-	-	-	D_0	D_1	D_2	D_3

Tabla 3.2: Inicialización de los agentes según que el sistema sea síncrono o asíncrono.

Para analizar el comportamiento del sistema recurriremos a medir la entropía de la información. En el apéndice A se puede consultar una descripción acerca de la entropía de acuerdo con la Teoría de la Información.

Resultados de la simulación

Los resultados de la evolución pueden observarse en las figuras 3.8 y 3.9.

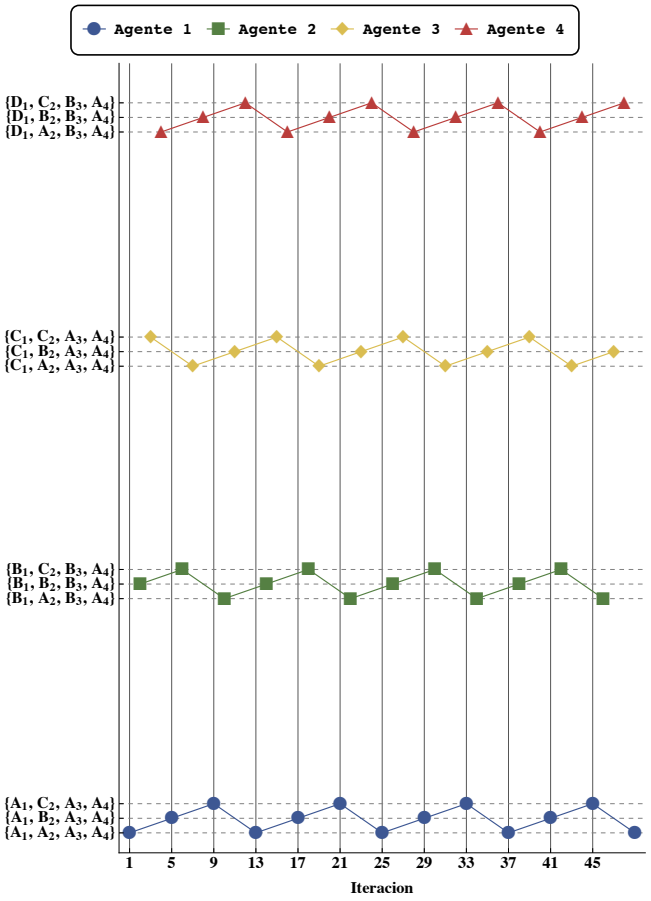


Figura 3.8: Evolución de los agentes para el modelo asíncrono.

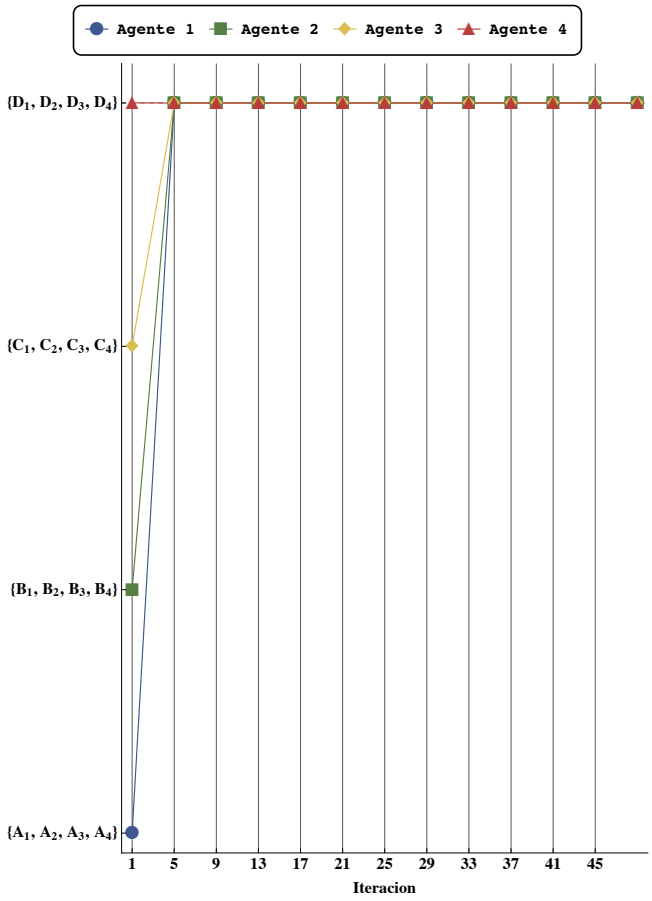


Figura 3.9: Evolución de los agentes para el modelo síncrono.

Como era de esperar, el modelo síncrono converge a una única secuencia. Esto se debe a que tras el estado inicial, donde cada agente genera una secuencia distinta, se actualiza la información por sobre-escritura, con lo cual el estado de dicha información coincide con el estado del último agente que ha actualizado la tabla: agente *D*.

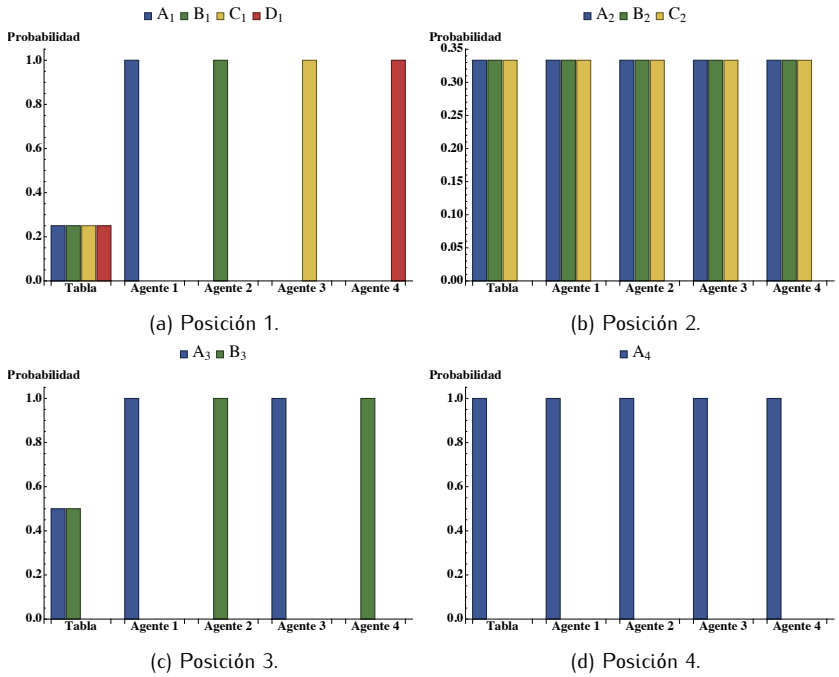
El caso del modelo asíncrono es mucho más interesante. La convergencia de cada agente es a una trayectoria periódica de 3 secuencias, donde cada secuencia se genera con la misma periodicidad: 4 iteraciones pero desfasadas.

Como se puede apreciar existe un componente temporal en el sistema asíncrono, dependiendo de la iteración, el contenido de información variará según el agente que en dicha iteración haya actualizado la tabla. Por el contrario, en el sistema síncrono esta componente no existe, independientemente de la iteración, el contenido de la tabla será siempre el mismo.

De hecho, se puede observar que los agentes generan secuencias pertenecientes a diferentes patrones, ver tabla 3.3. Igualmente podemos obtener las distribuciones de probabilidad de utilizar cierto símbolo en cierta posición de la secuencia. Estas distribuciones, tanto para los agentes como para la tabla, se muestran en la figura 3.10.

Agente	Posición 1	Posición 2	Posición 3	Posición 4
A	A_1	X	A_3	A_4
B	B_1	X	B_3	A_4
C	C_1	X	A_3	A_4
D	D_1	X	B_3	A_4

Tabla 3.3: Patrones de secuencias generados por cada agentes. El término en X indica que puede aparecer un símbolo A_2, B_2, o C_2.



(a) Posición 1.

(b) Posición 2.

(c) Posición 3.

(d) Posición 4.

Figura 3.10: Modelo asíncrono: distribución de probabilidad del uso de cada posible símbolo según la posición de la secuencia.

Con las distribuciones de probabilidad obtenidas, se puede obtener la entropía normalizada (acotada entre 0 y 1), del sistema y los agentes. Puesto que el contenido de información de los agentes es el mismo para todos ellos, basta con obtener la cantidad de información presente en uno de ellos. Los resultados del análisis de la entropía se muestran en la figura 3.11. Debido a que el número de símbolos utilizados decrece a medida que aumentamos las posiciones en una secuencia, la entropía también decrece a medida que avanzamos en las posiciones de una secuencia.

También es interesante remarcar que el contenido de información del sistema es superior al contenido de información de los agentes, sin llegar a ser la suma de todos ellos.

Antes de concluir la discusión del esquema, vamos a realizar un último análisis relativo a cómo la información está distribuida.

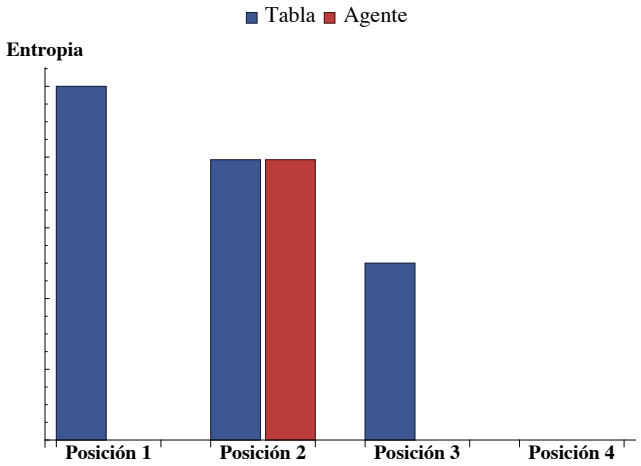


Figura 3.11: Contenido de información, entropía, respecto de cada posición en una secuencia.

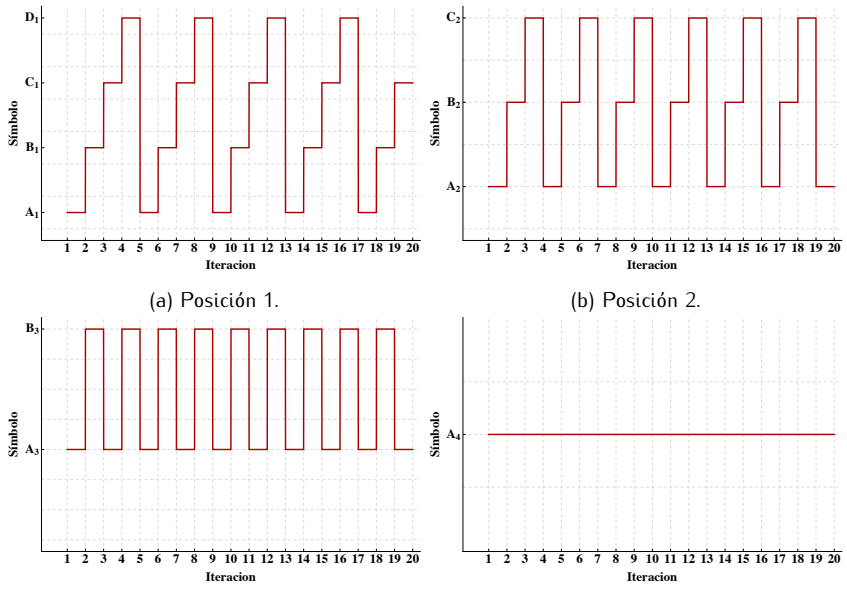
Distribución de la información

Podemos comenzar el análisis con un hecho muy simple, pero que hasta ahora no hemos tenido en cuenta: en un modelo inspirado en una colmena de abejas, la información se almacena en los agentes, por tanto en el mejor de los casos se puede tener información acerca de 4 secuencias distintas. En un modelo inspirado en las hormigas la información reside en la tabla, en nuestro caso 1 sola secuencia. El esquema propuesto almacena hasta 12 secuencias. La cuestión es: ¿cómo se distribuye la información? porque obviamente se almacena, se repite de manera periódica, y es superior a la capacidad de almacenamiento de las estructuras de memoria disponibles ($4 + 1 = 5$ secuencias).

La respuesta viene dada si atendemos a las propiedades de cada modelo biológico utilizado. Las abejas aportan un componente temporal, por otro lado, las hormigas añaden un componente local a la información. De la combinación de ambos componentes resulta un sistema donde la información se almacena en fragmentos que varían a diferente frecuencia lo que da lugar a una combinatoria que permite almacenar un número superior de secuencias.

Esto se ve muy claramente si analizamos la frecuencia de aparición de un determinado símbolo según la posición en la tabla. La figura 3.12 muestra este análisis, para cada posición, la señal es distinta porque el número de símbolos varía. Para la posición 1 la señal se repite cada 4 iteraciones, para la 2 cada 3 iteraciones, etc... Si solapásemos todas las señales nos daría lugar a un conjunto de combinaciones finitas de secuencias que son las que se observan en la evolución de la tabla del sistema.

Al combinar inspiración de ambos modelos, obtenemos un almacenamiento en frecuencia típico de las abejas sumado a una fragmentación de la información en función del estado del agente típica de las hormigas. Por esta razón se consigue aumentar la capacidad de almacenamiento de información del sistema.



(a) Posición 1.

(b) Posición 2.

(c) Posición 3.

(d) Posición 4.

Figura 3.12: Frecuencia de aparición de los símbolos para cada posición en la tabla del sistema asíncrono.

Organización de los agentes como una red

Para comprender mejor el funcionamiento del esquema que estamos utilizando podemos visualizarlo como una red. El concepto de red hay que entenderlo bajo la perspectiva de las denominadas redes complejas y teoría de redes [Mit09, Wat99]. Es otra forma de estudiar el comportamiento global de un sistema en función de las relaciones que se dan entre sus partes. Esta disciplina se basa en el uso de grafos para modelar el sistema, los nodos representan los componentes del sistema y las aristas las relaciones entre las partes.

El mismo enfoque de teoría de redes lo podemos aplicar en el estudio de nuestro esquema. Para analizar dicha red, es necesario atender a cómo circula la información entre los agentes y construir un modelo utilizando un grafo. Los nodos de dicho grafo serán los agentes del sistema y las aristas, información que se comunica.

El grafo resultante se ilustra en la figura 3.13. En cada color están marcados distintos sub-grafos, ya que la información no fluye por toda la red al mismo tiempo sino por etapas. Cuando el agente A actualiza la tabla, se activa la parte de la red correspondiente a la circulación de información entre el agente A y el resto (sub-grafo marcado en azul). Esto significa que la actividad de la red sigue un patrón: los distintos sub-grafos se van activando de manera consecutiva. Cada 4 ciclos se repite el mismo patrón.

Supongamos que se activa el patrón del agente B, el cual transmite el símbolo B_3 al agente D. El estado del agente B sería $[B_1]$. A continuación se activa el patrón del agente C, el cual transmite un símbolo X, el que sea, al agente B, que lo recoge en segunda posición: $[B_1, X_2]$. A continuación se activa el agente D que transmite el símbolo B_3 al agente B: $[B_1, X_2, B_3]$. Es decir, se establece un

patrón de comunicación entre el agente B y el D donde la información que circula entre ellos se transmite cada 2 ciclos. Este patrón permite conservar el símbolo B_3 como tercer elemento de una secuencia. Lo interesante es que esta información se almacena en ambos agentes por "turnos".

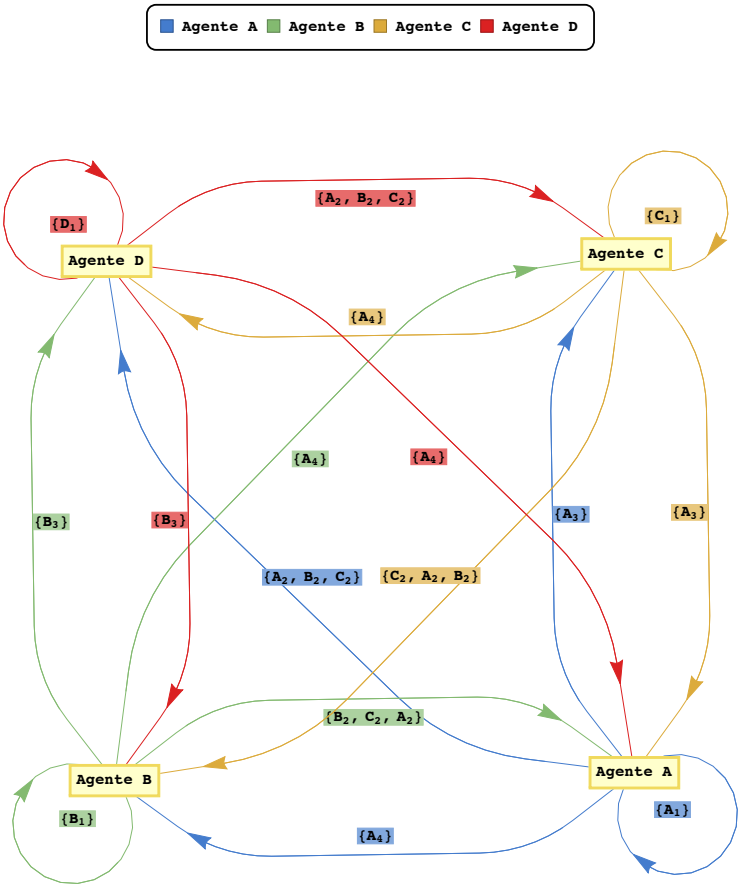


Figura 3.13: Red de comunicación que se establece entre los agentes del sistema. Los valores de las aristas representan la información transmitida y en qué posición se recoge.

Como se puede ver en el grafo, figura 3.13, distintos fragmentos de información recorren caminos distintos en la red, por tanto circulan a distinta velocidad. Por ejemplo ya hemos visto que el símbolo B_3 circula, es transmitido, cada dos ciclos. En cambio el símbolo A_1 únicamente viaja del agente A a sí mismo, como este agente transmite la información cada 4 ciclos, la velocidad de transmisión del símbolo A_1 es de cuatro ciclos. La distinta velocidad de circulación es la razón por la cual la red es capaz de almacenar información por encima de las capacidades de las estructuras de memoria del sistema. Si para cada posición de una secuencia, la información que se transmite y recoge en dicha posición circula a diferente

velocidad, es posible realizar un almacenamiento del conjunto de secuencias "codificadas en frecuencia".

Conclusión

A la vista de estas simulaciones se puede ver claramente que existe un componente temporal como el anteriormente visto en las colonias de abejas. Este componente temporal implica que la información ya no está contenida en la tabla, como ocurre en ACO, sino que se distribuye entre los agentes. Como todos los agentes no se encuentran en el mismo estado, la información que un agente comunica al resto únicamente está disponible durante una ventana de tiempo, hasta que el siguiente agente comunica la suya. En un esquema síncrono esta ventana no existe, al ser escrituras sincronizadas, la misma información está disponible durante toda la búsqueda de un agente.

El componente local de la información es algo que ya está presente en ACO. La novedad radica en el uso de la tabla para incorporar el componente temporal típico de las colmenas de abejas. Esto permite diversificar la búsqueda sin ralentizar la convergencia. Distintos grupos de agentes pueden trabajar de manera aislada con su propia información, haciendo posible la convergencia a soluciones distintas; lo cual reduce el estancamiento general del algoritmo.

En el modelo biológico de las abejas dejamos sin comentar el papel que juega la "respuesta a un umbral". Este umbral es necesario para evitar la dispersión en la toma de decisiones, ya que las fuentes de néctar que no son adecuadas (se encuentran por debajo del umbral) se abandonan, permitiendo que el proceso de decisión converja. Si no existiese este umbral, podría darse el caso de que la colonia explotase múltiples fuentes muy pobres y la recolección fuese ineficiente. Aunque en el esquema no lo hemos tenido en cuenta, en la implementación del prototipo necesitamos introducir un criterio de umbral para favorecer la convergencia del algoritmo. Dicho criterio se introducirá más adelante.

3.5. Implementación de un prototipo de algoritmo

Comenzaremos introduciendo los elementos que componen el prototipo, los agentes, las fuentes de información, etc... Una vez introducidos los elementos se pasará a introducir el esquema general y los procedimientos del sistema.

Para la implementación asumimos un esquema de minimización de costes. Para un esquema de maximización el desarrollo sería equivalente.

Definición de elementos

Recordemos brevemente que el algoritmo se utiliza un enfoque clásico de la Inteligencia Artificial: representar un problema en un espacio de estados. Los tres elementos fundamentales de dicha representación los constituyen el conjunto de estados (Q), el conjunto de acciones u operadores (U) y la función de transición ($\delta : Q \times U \to Q$).

Agente (A): dispone de dos listas, una que indica los estados recorridos y otra indicando la acción que se ha aplicado en cada estado.

Lista de estados (A_Q): $\rightarrow [q_0, q_1, ..., q_n] \quad q_0 \in Q_0 \wedge \forall i \in \{1, ...n\}, q_i \in Q$
Lista de acciones (A_U): $\rightarrow [u_0, u_1, ..., u_{n-1}] \quad \forall i \in \{0, ...n-1\}, u_i \in U$

La lista de estados comienza con un estado inicial (q_0), a continuación se disponen el resto de estados que ha atravesado el agente. El estado inicial al ser dado, implica que la lista de acciones contiene un elemento menos que el número de estados visitados. El último elemento de la lista de estados visitados indica el estado actual del agente.

Para cada estado q_i existe una acción asociada u_i que es la responsable de la transición al estado siguiente:

$$\delta(q_i, u_i) = q_{i+1}$$

Los agentes parten de la configuración inicial siguiente:

$$A_Q = [q_0]$$
$$A_U = []$$

Los agentes pueden ser de dos tipos:

a) Patrollers, agentes que al buscar adquieren nueva información.

b) Foragers, agentes que buscan utilizando únicamente la información adquirida por el algoritmo.

El hecho de que haya dos agentes en el sistema guarda relación con el balance de la búsqueda. Este balance lo podemos conseguir mediante el equilibrio de una población donde cada agente esté asignado a una tarea distinta: explorar o explotar. A medida que avancemos en el desarrollo del estudio se entenderá mejor el papel de cada agente.

Heurística (η): representa la información previa, viene definida como una distribución de pesos, mediante la cual se obtiene una distribución de probabilidad utilizando una función de normalización:

$$\eta : Q \times U \rightarrow \{x \in \mathbb{R} | 0 \leq x \leq 1\}$$

la función recibe como entrada un estado y una acción y devuelve un valor de probabilidad.

A través de la fuente de información previa se puede incorporar el conocimiento a priori que dispongamos de un problema: la heurística.

En caso de no disponer de conocimiento heurístico, simplemente la podemos definir como una función constante, asignando el mismo peso a toda acción en cualquier estado, dando lugar a un distribución uniforme.

Feromona (τ): representa la información adquirida. Igualmente viene dada por una distribución de pesos, la cual será necesario normalizar para obtener una distribución de probabilidad:

$$\tau : Q \times U \rightarrow \{x \in \mathbb{R} | 0 \leq x \leq 1\}$$

Es importante recalcar que al ser información adquirida, la función τ está parcialmente definida. El sistema adquiere información a partir de las búsquedas, por tanto sólo se dispondrá de entradas –de información– para aquellos pares estado-acción que hayan sido utilizados en alguna búsqueda.

En los casos donde no esté definida la feromona, los agentes deberán recurrir a la información heurística, la cual sí está definida para todo estado y acción.

Umbral (μ) el umbral es aquel valor de coste mínimo para considerar una búsqueda exitosa y su solución aceptable. En caso de que una búsqueda sea exitosa se procede a actualizar el contenido de la feromona (τ), de este modo el algoritmo adquiere conocimiento. En nuestro caso como criterio de umbral vamos a utilizar la media (μ) del coste de las soluciones conocidas hasta el momento.

El uso del umbral es idéntico al de una colonia de abejas, determinar cuando un agente comunica su información al resto, haciendo pública dicha información. En nuestro caso la información se hace pública, se comunica, incorporando dicha información al contenido de la feromona.

Una vez introducidos los elementos del sistema, pasamos a detallar los procedimientos.

Esquema general

El sistema es un algoritmo de búsqueda iterativo, es decir, se compone de un bucle principal, dentro del cual podemos distinguir las dos etapas mostradas en la figura 3.14.

Construcción de soluciones: engloba aquellos procedimientos necesarios para que un agente construya una solución –una secuencia–. Al ser un esquema constructivo, las secuencias se construyen paso a paso, cada uno constituye lo que denominamos paso de búsqueda.

En cada *paso de búsqueda* el agente debe realizar tres procedimientos:

a) Selección de una fuente de información: los agentes disponen de dos fuentes de información, deben elegir cual de ellas usar para llevar a cabo el paso de búsqueda actual. No se permite el uso combinado de ambas fuentes en un mismo paso de búsqueda.

b) Selección de una acción: una vez seleccionada una fuente de información, el agente debe escoger la acción que va a realizar.

c) Actualización del estado: por último el agente debe aplicar la acción seleccionada para obtener un nuevo estado. Ademas, el agente actualiza la lista de estados visitados y acciones aplicadas.

El esquema general de un paso de búsqueda por parte de un agente puede verse en el pseudo-código que describe al algoritmo, figura 3.14.

Actualización de información: engloba aquellos procedimientos necesarios para que un agente actualice el contenido de la feromona (información adquirida) y el valor de umbral de acuerdo con la secuencia construida.

El primer paso es considerar si la búsqueda del agente ha tenido éxito o no, para lo cual se utilizará un criterio de umbral. En caso de que al agente haya tenido éxito, se actualizará la información. En caso contrario el agente no hará nada (se descarta la información de la búsqueda).

El esquema general de la actualización puede verse en el pseudo-código que describe al algoritmo, figura 3.14. El procedimiento $finalizado()$ representa el criterio de parada para determinar si la búsqueda del agente ha finalizado, El término $J(S_a)$ representa el coste dado por la función objetivo del problema(J asociado a la solución del agente: S_a.

```
1:  Inicialización
2:  while condición de terminación no satisfecha do
3:      for a ∈ Población do                          ▷ Paso de búsqueda
4:          seleccionar información
5:          escoger acción
6:          actualizar(a)
7:      end for
8:      for a ∈ Población do                  ▷ Actualización de información
9:          if finalizado(a) == ⊤ then
10:             if J(S_a) < μ then                        ▷ Criterio de umbral
11:                 actualizar(μ, a)
12:                 actualizar(τ, a)
13:             end if
14:         end if
15:     end for
16: end while
```

Figura 3.14: Esquema general en pseudo-código.

Construcción de soluciones

La construcción de soluciones se detalla en la figura 3.15.

- En primer término el agente selecciona una lista de acciones disponibles (L_U), donde cada acción tiene una cierta probabilidad de ser escogida: $P(L_U)$.
- Como existen dos fuentes de información, es necesaria una política de control que regule cuando se utilizará la heurística o la feromona para determinar los valores de L_U y $P(L_U)$.
- A continuación, el agente escoge un acción perteneciente a L_U haciendo uso de la distribución de probabilidad asociada: $P(L_U)$.

- Por último el agente actualiza su estado actual, la lista de estado visitados y las acciones aplicadas.

```
1: procedure Paso de búsqueda(agente)
2:     {L_U, P(L_U)} ← política de control
3:     acción ← selección(L_U, P(L_U))
4:     agente ← actualizar(agente)
5: end procedure
```

Figura 3.15: Construcción de soluciones: paso de búsqueda.

Comenzaremos definiendo *un paso de búsqueda* que consiste en la actualización de las estructuras de datos de un agente.

Política de control

La finalidad de la política de control es determinar los valores de L_U y $P(L_U)$ en función de la información que se utilice. Antes de desarrollar un criterio de decisión, veremos como obtener estos valores dependiendo de la información.

Feromona (τ): representa la información adquirida que se codifica utilizando una tabla. Siguiendo la nomenclatura de ACO, la denominaremos *tabla de feromona*.

Se trata de una *tabla asociativa*, la cual almacena la información referente a los estados y las acciones. Una tabla asociativa es una estructura que permite el acceso directo al contenido en función de una clave. En este caso las claves son estados, y el contenido una lista de pares acción-peso. La estructura de la tabla se representa en la figura 3.16.

Clave	Valor
q_0	$[(u_0, \tau_0), (u_1, \tau_1), \ldots]$
q_1	$[(u_1, \tau_1), (u_2, \tau_2), \ldots]$

Figura 3.16: Estructura de la tabla de feromona. El símbolo τ_i representa un peso.

Utilizando la tabla, la probabilidad asociada a una acción en función de un estado queda definida mediante una sencilla ecuación de normalización:

$$P_\tau(u_i|q_n) = \frac{\tau_{u_i}^{q_n}}{\sum_{j \in |\tau^{q_n}|} \tau_{u_j}^{q_n}} \tag{3.1}$$

donde $\tau_{u_i}^{q_n}$ es el peso asociado a la acción u_i correspondiente al estado q_n, y $|\tau^{q_n}|$ es el número de elementos de la lista asociada al estado q_n.

- Para un cierto estado q_n definimos la lista de acciones disponibles ($L_U^{q_n}$) como aquellas acciones que estén contenidas en la lista de la tabla asociada al estado q_n:

$$L_U^{q_n} = [u_i] \quad u_i \in \tau^{q_n}$$

- Una vez que está definida la lista de acciones disponibles ($L_U^{q_n}$), la probabilidad de cada acción viene dada por la ecuación 3.1 :

$$P(L_U^{q_n}) = [P_\tau(u_i|q_n)] \quad u_i \in L_U^{q_n}$$

Heurística (η): representa la información previa que viene dada por una función dependiente del problema (η). A su vez se asume que existe otra función ad-hoc para cada problema ($\Upsilon : Q \to U$), la cual dado un estado, devuelve una lista de acciones disponibles para dicho estado.

Utilizando la función η y la función Υ, la probabilidad asociada a una acción en función de un estado queda definida mediante otra ecuación de normalización:

$$P_\eta(u_i|q_n) = \frac{\eta(u_i, q_n)}{\sum_{j \in |\Upsilon(q_n)|} \eta(u_j, q_n)} \tag{3.2}$$

donde $\eta(u_i, q_n)$ es el peso asociado a la acción u_i correspondiente al estado q_n, y $|\Upsilon(q_n)|$ es el número de acciones disponibles para el estado q_n.

- Para un cierto estado q_n, la lista de acciones disponibles vendrá dada por un procedimiento o función, representado por el término Υ, que será dependiente de la definición del problema:

$$L_U^{q_n} = \Upsilon(q_n)$$

- Una vez que está definida la lista de acciones disponibles ($L_U^{q_n}$), la probabilidad de cada acción viene dada por la ecuación 3.2 :

$$P(L_U^{q_n}) = [P_\eta(u_i|q_n)] \quad u_i \in L_U^{q_n}$$

Selección de información según el tipo de agente

Una vez que tenemos detallado como obtener L_U y $P(L_U)$ podemos pasar a detallar el criterio que determina cuándo utilizar la feromona o la heurística.

Para indicar el uso de la feromona se recurrirá a los símbolos L_U^τ y $P(L_U^\tau)$, mientras que para denotar la heurística se utilizaran los símbolos L_U^η y $P(L_U^\eta)$.

Foragers. Este tipo de agente trata de usar exclusivamente la información adquirida, luego el criterio es muy sencillo: si existe información en la tabla de feromona para el estado actual del agente se hará uso de ella. En caso de no existir dicha información, se recurre a la información heurística.

Más formalmente:

$$\begin{cases} L_U^\tau = \emptyset & \to L_U^\eta \wedge P(L_U^\eta) \\ \text{e.o.c} & \to L_U^\tau \wedge P(L_U^\tau) \end{cases} \tag{3.3}$$

La razón de recurrir a la información heurística en caso de no existir información adquirida tiene como motivo el preservar la búsqueda que ha realizado el agente.

Patrollers. En este caso el agente debe hacer uso de la heurística para adquirir nueva información e incorporarla a la tabla de feromona.

Cuando un patroller debe realizar una acción, escoge entre usar la heurística o la feromona, pero nunca combina ambas fuentes de información. El criterio que regula el uso de una u otra es el siguiente:

- Si la última información utilizada provenía de la feromona (L_U^τ), o el estado actual es un estado inicial, entonces:

$$\begin{cases} \chi < 1 - (1 - \gamma_1)^{1/NS} & \rightarrow L_U^\eta \wedge P(L_U^\eta) \\ \text{e.o.c} & \rightarrow \text{eq. 3.3} \end{cases} \tag{3.4}$$

- Si la última información utilizada provenía de la heurística (L_U^η):

$$\begin{cases} \chi < \gamma_2 & \rightarrow L_U^\eta \wedge P(L_U^\eta) \\ \text{e.o.c} & \rightarrow \text{eq. 3.3} \end{cases} \tag{3.5}$$

Como se puede observar el criterio es más complejo que en el caso anterior, lo explicamos por partes:

- El término $\rightarrow$ eq. 3.3 simplemente indica que cuando un patroller deba usar la feromona pero no esté definida para el estado actual, entonces recurrirá a la heurística.

- El término $\chi < 1 - (1 - \gamma_1)^{1/NS}$ regula el uso de la heurística empleando $\chi \in (0, 1]$, un número generado al azar. El término $\gamma_1 \in [0, 1)$ es un parámetro definido por el usuario y NS es el número de acciones necesarias para construir una secuencia completa[1].

  En la figura 3.17 se muestra el valor de χ para distintos valores de NS y γ_1. Como puede verse hasta que γ_1 no se aproxima a valores cercanos a 1, el valor de χ es próximo a 0. A medida que aumentamos el valor de NS aumenta la influencia de la raíz, impidiendo el crecimiento de la curva hasta que γ_1 no alcanza valores realmente próximos a 1.

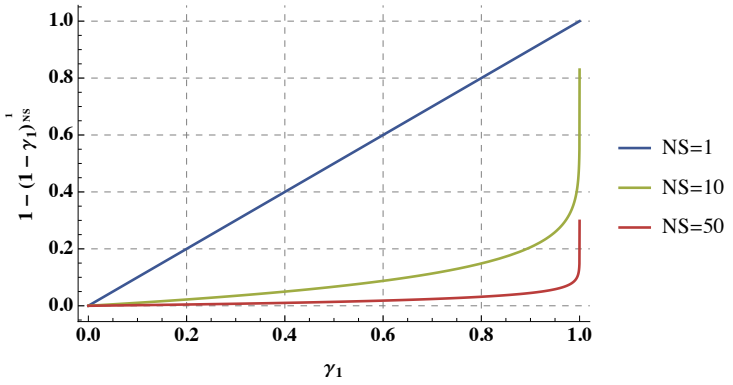


Figura 3.17: Evaluación de la expresión $1 - (1 - \gamma_1)^{1/NS}$ para distintos valores de NS y γ_1.

[1] Puede darse el caso que el número de acciones sea variable, de ser así, para el valor de NS habrá que recurrir a algún método de estimación sencillo. Como usar la longitud de la mejor solución conocida, etc...

El parámetro γ_1 refleja la probabilidad de que un patroller utilice *al menos una vez* la información heurística para construir una secuencia. Ya que si esto no ocurriese, el patroller no adquiriría información nueva. No habría diferencia entre un forager y un patroller.

Supongamos que queremos construir una secuencia de n elementos, usando una probabilidad p de utilizar la heurística. Por tanto, la probabilidad de construir una secuencia utilizando al menos una vez la heurística viene dada por la siguiente expresión: $1-(1-p)^n$.

Esta última expresión nos permite relacionar el valor de p con γ_1:

$$\gamma_1 = 1-(1-p)^n$$
$$p = 1-\sqrt[n]{1-\gamma_1}$$

Intuitivamente manejar γ_1 es mucho más simple que asignar un valor a p. Porque, independientemente del problema, los patrollers deberán construir una secuencia utilizando la información previa *al menos una vez*. Por tanto γ_1 debe reflejar una probabilidad alta (≈ 1).

- El término $\chi < \gamma_2$ introduce el uso condicional de un segundo parámetro $\gamma_2 \in [0,1)$. De esta manera permite la definición de una probabilidad recurrente[2], donde la decisión actual está condicionada en función del tipo de información utilizada en la decisión anterior. Es decir, la probabilidad (p) de que una acción se escoja utilizando la heurística depende de la probabilidad anterior:

$$p[0] = p_1$$
$$p[n] = p_1 + (p_2 - p_1)\cdot p[n-1]$$

donde $p_1 = 1-\sqrt[NS]{1-\gamma_1}$ y $p_2 = \gamma_2$.

Resolviendo la recurrencia, obtenemos la siguiente expresión general:

$$p[n] = \frac{p_1 - p_1\cdot(p_2-p_1)^n}{1+p_1-p_2}$$

Como tanto p_1 como p_2 son valores positivos menores que 1, existe un límite en la recurrencia:

$$\lim_{n\to\infty} \frac{p_1 - p_1\cdot(p_2-p_1)^n}{1+p_1-p_2} = \frac{p_1}{1+p_1-p_2}$$

Si evaluamos la expresión $p[n]$, figura 3.18, para unos ciertos valores ($\gamma_1 = 0{,}99$, $\gamma_2 = 0{,}5$, $NS = 10$), podemos ver como la probabilidad crecer hasta que tiende a un límite: comienza a saturar alcanzado un valor estable.

El hecho de introducir un segundo parámetro γ_2 permite definir una probabilidad que aumenta a medida que se toman más decisiones, favoreciendo el uso de la feromona en las primeras decisiones donde se puede asumir que

[2]Una ecuación recurrente (o ecuación en diferencias) es el análogo discreto de una ecuación diferencial.

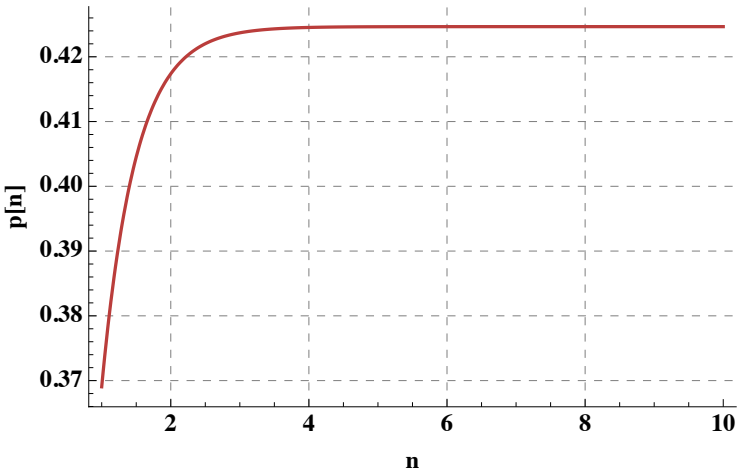


Figura 3.18: Evolución de la probabilidad ($p[n]$) de utilizar la información heurística para seleccionar una acción en función del número previo de pasos de búsqueda dados (n).

es más fiable. Este uso decrece a medida que tomamos más decisiones y la fiabilidad sea dudosa.

Selección de una acción

Este procedimiento es una selección al azar sobre una lista de acciones L_U, donde cada acción tiene una cierta probabilidad de ser escogida, $P(L_U)$.

Una selección al azar consiste en escoger la primera acción cuya probabilidad acumulada sea mayor que un cierto valor χ generado utilizando una distribución uniforme entre $(0, 1]$.

Formalmente, la selección al azar de una acción la podemos definir de la siguiente manera:

Sean

$$L_U = [u_0, u_1, ..., u_{n-1}, u_n]$$
$$P(L_U) = [p(u_0), p(u_1), ..., p(u_{n-1}), p(u_n)]$$

La acción seleccionada u_i vendrá dada por el mínimo valor de i:

$$\min_i \quad i \in \{0, 1, ..., n\}$$
$$\chi < F(u_i)$$

- χ es un número aleatorio generado a partir de una distribución uniforme entre $(0, 1]$
- $F(u_i) = \sum_{j=0}^{i} p(u_j)$

Una vez que el agente ha escogido una acción, únicamente resta actualizar su estado.

Actualización del agente

Este procedimiento consiste en aplicar una acción determinada u_n para transitar a un nuevo estado. La lista de estados visitados se actualiza de acuerdo al estado resultante de la transición y la acción utilizada se añade a la lista de acciones aplicadas. Formalmente podemos definir el procedimiento de la siguiente manera:

Dado un agente A tal que

$$A_Q = [q_0, q_1, ..., q_n]$$
$$A_U = [u_0, u_1, ..., u_{n-1}]$$

la aplicación de la acción u_n por parte del agente da como resultado la siguiente actualización:

$$A_Q^+ = [q_0, q_1, ..., q_n] +\!\!+ [q_{n+1}] = [q_0, q_1, ..., q_n, q_{n+1}]$$
$$A_U^+ = [u_0, u_1, ..., u_{n-1}] +\!\!+ [u_n] = [u_0, u_1, ..., u_{n-1}, u_n]$$

donde $q_{n+1} = \delta(q_n, u_n)$.

Actualización de la información

La actualización de la información es el procedimiento mediante el cual un agente modifica el contenido de información de la tabla de feromona, la información adquirida, en función de la solución que ha construido.

Este procedimiento de actualización se compone de dos pasos:

a) Actualización del umbral

b) Actualización de la tabla de feromona

Detallaremos cada paso de manera independiente.

Actualización del umbral

En nuestro caso como valor umbral vamos a utilizar la media (μ) del coste de las soluciones conocidas hasta el momento. El utilizar la media tiene varias ventajas fundamentales:

i) Es un valor definido completamente por el estado de la búsqueda. No requiere configuración por parte del usuario y además no introduce ningún sesgo en la dinámica del sistema, lo cual es fundamental cuando se trabaja con sistemas auto-organizados.

   Un sistema auto-organizado se comporta en función de su entorno, de su marco de trabajo. En el caso de un algoritmo el entorno es el problema dado. El hecho de introducir elementos previamente definidos que no forman parte del problema, pero sí del marco de trabajo, por ejemplo parámetros definidos

por el usuario, implica que el sistema se va a organizar de acuerdo al marco provisto, no al problema. De tal modo que la organización resultante puede no ser adecuada para resolver el problema, simplemente porque el marco de trabajo definido no es correcto.

ii) La media tiene un comportamiento asintótico en función del número de muestras. Si además únicamente actualizamos la media con valores inferiores a ella, esto implica que la media sólo puede decrecer. El resultado es que su valor decrecerá rápidamente si el número de muestras es pequeño y tenderá a permanecer estable si el número es alto.

Trasladado a un proceso de búsqueda implica que el umbral variará significativamente en los primeros estadios de la búsqueda, lo cual es beneficioso porque limita la ganancia de información e impide la dispersión en esas primeras iteraciones. Pero a su vez tiende a estabilizarse a medida que el proceso avanza, favoreciendo la ganancia de información y la dispersión a medida que la búsqueda avanza.

iii) El cálculo de la media es un proceso computacionalmente eficiente, únicamente requiere operaciones aritméticas sencillas:

$$\mu_{n+1} = \mu_n + \frac{x_n - \mu_n}{n+1}$$

De manera genérica podemos definir el umbral como una media móvil gobernada por un parámetro γ_3 que limita el número de muestras empleado en el cálculo de la media. En general, dicho parámetro se puede asignar como infinito, $\gamma_3 = \infty$, de este modo la media móvil sería simplemente la media "clásica": el valor ponderado de todas las soluciones conocidas.

El procedimiento de actualización de la media para $\gamma_3 = \infty$ se detalla en la figura 3.19. El procedimiento es muy simple: se lleva un contador con el número de muestras m y se actualiza el valor de la media de la manera correspondiente.

La única salvedad es el caso inicial: al tratarse de un umbral, antes de obtener la primera solución, la media tiene valor ∞. De este modo siempre se acepta la primera solución proveniente de una búsqueda. Cuando la media se actualiza por primera vez, se inicializa al valor del coste de esa primera solución.

```
1: procedure ACTUALIZA MEDIA(S, m)
2:     m ← m + 1
3:     if m = 1 then                          ▷ Inicialización de la media
4:         μ ← J(S)
5:     else
6:         μ ← μ + (J(S) − μ)/m
7:     end if
8: end procedure
```

Figura 3.19: Actualización de la media: descripción en pseudo-código, para $\gamma_3 = \infty$. El símbolo S indica una solución y m el número de muestras.

En la figura 3.20 se detalla el procedimiento en caso de que $\gamma_3 \neq \infty$. El procedimiento es prácticamente igual, únicamente se incluye una pila L_μ donde se almacenan los valores sobre los que se obtiene la media. Primeramente vamos añadiendo valores a la pila hasta que se alcanza el número de muestras permitido. Una vez que se alcanza dicho límite, se extrae el último elemento de la pila ($L_\mu(\gamma_3)$), se añade el nuevo valor y se actualiza la media de la manera correspondiente.

```
1:  procedure ACTUALIZA MEDIA(S, m)
2:      m ← m + 1
3:      L_μ ← add(J(S))
4:      if m = 1 then
5:          μ ← J(S)                          ▷ Inicialización de la media
6:      else
7:          if m < γ_3 then
8:              μ ← μ + (J(S) − μ)/m
9:          else
10:             μ ← μ + J(S)/γ_3 − L_μ(γ_3)/γ_3
11:             L_μ ← extraer(γ_3)
12:             m ← m − 1
13:         end if
14:     end if
15: end procedure
```

Figura 3.20: Actualización de la media: descripción en pseudo-código. El símbolo S indica una solución, m el número de muestras y L_μ es una pila donde se almacenan los valores sobre los que se obtiene la media.

Actualización de la tabla de feromona

Cuando introdujimos el esquema híbrido (sección 3.4), describimos la política de actualización como una operación de sobre-escritura. Obviamente para un algoritmo necesitamos un procedimiento más refinado. Lo que vamos a hacer es ponderar el reemplazo de la información contenida en la tabla por la información proveniente de una búsqueda en función de la calidad de la solución encontrada.

Elementos del procedimiento

Previamente a describir el procedimiento de actualización, vamos a definir los elementos que lo conforman.

La calidad de una solución (λ) se indica mediante un número entre 0 y 1 que se calcula tomando como referencias el valor de umbral μ y el coste de la mejor solución encontrada hasta el momento (J_{best}):

$$\lambda = \frac{J(S) - \mu}{J_{best} - \mu} \tag{3.6}$$

donde $J(S)$ es el coste asociado a la solución generada por el agente.

Es decir, una solución S tendrá mayor calidad a medida que su coste se acerque al coste de la mejor solución descubierta hasta el momento (J_{best}), y calidad 0 si su coste es igual o inferior al valor umbral μ que indica si una solución es aceptable o no.

Frecuencia relativa (ν) de una acción respecto a un estado, es el número de veces que el agente A ha utilizado dicha acción en dicho estado, respecto al número de veces que el agente ha visitado dicho estado. Más formalmente:

$$\nu_{u_i}^{q_n} = \frac{N_{u_i}^{q_n}}{N_{q_n}} \tag{3.7}$$

donde $N_{u_i}^{q_n}$ indica el número de veces que el agente ha aplicado la acción u_i en el estado q_n, N_{q_n} es el número de veces que el agente ha visitado el estado q_n.

Por ejemplo:

$$A_j = \left[A_Q = [q_m, q_n, q_m, q_n, q_m], A_U = [u_i, u_j, u_j, u_i, u_i]\right]$$
$$\nu_{u_i}^{q_m} = \frac{2}{3} \quad \nu_{u_i}^{q_n} = \frac{1}{2}$$
$$\nu_{u_j}^{q_m} = \frac{1}{3} \quad \nu_{u_j}^{q_n} = \frac{1}{2}$$
$$\nu_{u_l}^{q_m} = 0 \quad \nu_{u_l}^{q_n} = 0$$

La ecuación que define como actualizar el valor de τ asociado a una acción en la tabla es la siguiente:

$$\tau^+ = \tau + (C - \tau) \cdot \lambda \tag{3.8}$$

La ecuación que estamos utilizando es de nuevo recurrente, donde el valor de τ se aproxima a C. El valor de λ gobierna la velocidad de convergencia.

Como λ es un número positivo menor que 1, si resolvemos la recurrencia para un valor inicial τ_0, podemos ver que al ir aumentando las actualizaciones ($n \to \infty$) el valor de τ será aproximadamente C, independientemente de λ y su valor inicial (τ_0).

$$\tau^+ = \tau + (C - \tau) \cdot \lambda$$
$$\tau n = C + (\tau_0 - C) \cdot (1 - \lambda)^n$$
$$\lim_{n \to \infty} C + (\tau_0 - C) \cdot (1 - \lambda)^n = C$$

Esta tendencia en el comportamiento de la ecuación 3.8 se ilustra en la figura 3.21.

Utilizando la frecuencia relativa, podemos particularizar la ecuación 3.8 de la siguiente manera:

$$\tau^+ = \tau + (\nu - \tau) \cdot \lambda \tag{3.9}$$

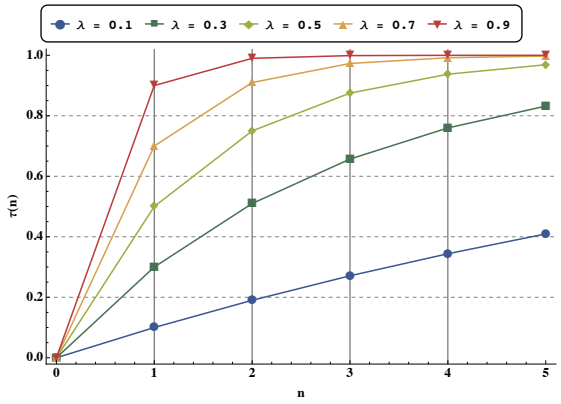


Figura 3.21: Evaluación de la ecuación $\tau n = C + (\tau_0 - C) \cdot (1 - \lambda)^n$ para $C = 1$ y $\tau_0 = 0$.

La ecuación 3.9 indica que el peso asociada a una acción se actualiza de manera que dicho valor refleje la frecuencia relativa de dicha acción.

Procedimiento de actualización

El procedimiento de actualización se detalla en la figura 3.22. Para cada estado visitado por el agente (A_q) se actualiza el contenido de la tabla ($\tau(q_i)$) modificando el valor asociada a cada acción en función de la calidad de la solución obtenida por el agente (λ). Como se puede observar, el valor asociado refleja la frecuencia relativa: la probabilidad de utilizar una determinada acción en un determinado estado.

El contenido de la tabla será reemplazado en mayor o menor medida en función del valor de λ. Es importante remarcar que este reemplazo puede darse porque λ tenga un valor próximo a 1. Pero también puede darse si se acumula el efecto de sucesivas actualizaciones con valores menores de λ. El contenido se conservará únicamente si la acción utilizada es siempre la misma.

```
1:  procedure ACTUALIZACIÓN(A_q, A_u, λ)
2:      for i ∈ |A_q| do                    ▷ Estados visitados por el agente
3:          q_i ← A_q(i)
4:          L_τ ← τ(q_i)                    ▷ Contenido de la tabla para el estado q_i
5:          for j ∈ |L_τ| do
6:              [τ_j, u_j] ← L_τ(j)
7:              τ_j = τ_j + (v_{u_j}^{q_i} − τ_j) · λ
8:          end for
9:      end for
10: end procedure
```

Figura 3.22: Actualización de la información: descripción en pseudo-código.

Un caso particular que ayuda a ilustrar el procedimiento de actualización se

da cuando **los estados únicamente pueden ser visitados una vez**. De hecho es una restricción muy típica en diversos problemas, con lo cual merece la pena ilustrar el procedimiento de actualización para este caso.

Si un estado sólo puede ser visitado una vez, entonces $v_{u_j}^{q_i}$ sólo puede ser 1 si la acción aplicada en el estado q_i es u_j. En caso contrario la frecuencia sería 0. De este modo el procedimiento de actualización sería el mostrado en la figura 3.23.

Para este caso particular puede observarse mejor el reemplazo de la información, puesto que si $\lambda = 1$, entonces, para los estados visitados por el agente, la tabla únicamente contiene las acciones que dicho agente ha utilizado. El resto de información ha sido descartada.

```
1:  procedure ACTUALIZACIÓN(A_q, A_u, λ)
2:      for i ∈ |A_q| do                    ▷ Estados visitados por el agente
3:          q_i ← A_q(i)
4:          L_τ ← τ(q_i)                    ▷ Contenido de la tabla para el estado q_i
5:          for j ∈ |L_τ| do
6:              [τ_j, u_j] ← L_τ(j)
7:              if u_j == A_u(i) then
8:                  τ_j = τ_j + (1 − τ_j) · λ
9:              else
10:                 τ_j = τ_j + (0 − τ_j) · λ
11:             end if
12:         end for
13:     end for
14: end procedure
```

Figura 3.23: Actualización de la información: descripción en pseudo-código para el caso de que los estados únicamente puedan ser visitados una vez.

Por último es interesante resaltar que la información no se modifica para aquellos estados que no han sido visitados por un agente: dicha información no se pierde con el tiempo, con lo cual, los agentes pueden re-utilizarla en cualquier momento de la búsqueda.

Esquema en pseudo-código

Una vez que hemos introducido todos los elementos que forman parte de la actualización de la información, podemos reunirlos en un único esquema para visualizar el orden en que se realiza cada procedimiento. El esquema se muestra en la figura 3.24, se han vuelto a incluir los procedimientos previamente descritos para la actualización de la tabla y del valor de la media (criterio de umbral).

```
1: procedure Bucle Principal
2:     for a ∈ Población do                          ▷ Actualización de información
3:         if finalizado(a) == ⊤ then
4:             S ← a(S)                  ▷ Obtenemos la solución a partir del agente
5:             if J(S) < μ then                                  ▷ Criterio de umbral
6:                 if J(S) < J_best then          ▷ Actualizamos la mejor solución
7:                     J_best ← J(S)
8:                 end if
9:                 μ ← actualizaMedia(S, m)
10:                λ ← (J(s)−μ)/(J_best−μ)          ▷ Calidad de la solución (Eq. 3.6).
11:                τ ← actualizaTabla(a(A_q), a(A_u), λ)
12:            end if
13:        end if
14:    end for
15: end procedure
16:
17: procedure actualizaMedia(S, m)
18:    m ← m + 1
19:    L_μ ← add(J(S))
20:    if m = 1 then
21:        μ ← J(S)                                  ▷ Inicialización de la media
22:    else
23:        if m < γ3 then
24:            μ ← μ + (J(S) − μ)/m
25:        else
26:            μ ← μ + J(S)/γ3 − L_μ(γ3)/γ3
27:            L_μ ← extraer(L_μ(γ3))
28:            m ← m − 1
29:        end if
30:    end if
31: end procedure
32:
33: procedure actualizaTabla(A_q, A_u, λ)
34:    for i ∈ |A_q| do                          ▷ Estados visitados por el agente
35:        q_i ← A_q(i)
36:        L_τ ← τ(q_i)                ▷ Contenido de la tabla para el estado q_i
37:        for j ∈ |L_τ| do
38:            [τ_j, u_j] ← L_τ(j)
39:            if u_j == A_u(i) then
40:                τ_j = τ_j + (1 − τ_j) · λ
41:            else
42:                τ_j = τ_j + (0 − τ_j) · λ
43:            end if
44:        end for
45:    end for
46: end procedure
```

Figura 3.24: Proceso de actualización de la información.

3.6. Análisis Experimental

Una vez introducida la implementación del prototipo podemos realizar un pequeño estudio experimental para su verificación. Para llevarlo a cabo se va a utilizar el problema del viajante de comercio: Travelling Salesman Problem (TSP) [JM97], consistente en recorrer n ciudades sin pasar dos veces por la misma, exceptuando la ciudad inicial, de manera que la distancia recorrida sea mínima. Las instancias que utilizaremos se toman de la librería TSPLIB'95 [Rei95], donde el número de ciudades viene indicadas en el nombre de la instancia.

El TSP es un problema bien conocido, que tiene una heurística muy fiable, definida como la inversa de la distancia entre dos ciudades. A su vez, el problema es cíclico, cada ciudad es inicio y final de un recorrido, permitiendo construir un mismo recorrido partiendo de distintas ciudades.

Para resolver el TSP utilizaremos una representación en espacio de estados muy sencilla: el estado actual viene determinado por la ciudad actual. Desde cada estado, las acciones disponible son ciudades que se pueden visitar, que no han sido visitadas con anterioridad. La solución final viene dada por la lista ordenada de ciudades visitadas. No se utilizará la heurística, sino que las ciudades se escogen al azar. Los recorridos se construyen siempre partiendo de la misma ciudad.

La representación que utilizamos no persigue resolver el TSP, sino analizar el prototipo bajo unas condiciones generales: ausencia de heurística y estado inicial fijo.

Para este pequeño estudio experimental utilizaremos dos variaciones del prototipo: una versión síncrona, donde todas las hormigas presentan siempre un número idéntico de ciudades visitadas. Esta versión se compara con una versión asíncrona, donde una hormiga puede llevar visitadas x ciudades y otra x'. El objetivo es comparar el rendimiento de ambos esquemas.

En todos los casos utilizaremos igual número de hormigas que ciudades tiene la instancia del TSP. La razón se debe a que éste es –aproximadamente– el valor necesario para que se observe con claridad la diferencia entre la versión asíncrona y la síncrona.

Tabla de feromona

Un primer experimento, consiste en estudiar la evolución de la tabla de feromona a lo largo de la búsqueda. Es interesante porque el contenido de la tabla influye sustancialmente en el rendimiento del algoritmo.

Para realizar el estudio hemos utilizado una instancia de 51 ciudades ($Eil51$), la población consiste en 51 hormigas. Hemos ido variando la configuración de la población, indicando el porcentaje de foragers que la componen, para ilustrar como influye cada tipo de agente. Para cada configuración de la población, se ha ejecutado el algoritmo 100 veces y se ha calculado un promedio de la entropía media de la tabla en función del número de soluciones construidas.

En la figura 3.25 se puede ver el resultado de este experimento. En el caso síncrono la entropía de la tabla desciende a cero drásticamente a no ser que la población no esté compuesta exclusivamente por patrollers. En el caso asíncrono, se puede observar como la entropía media es proporcional al porcentaje de foragers empleados: cuantos más foragers menos entropía y menos dispersión.

Cuando una hormiga actualiza la tabla, incrementa la probabilidad de las acciones utilizadas y decrementa el resto. Por tanto, a medida que se realizan

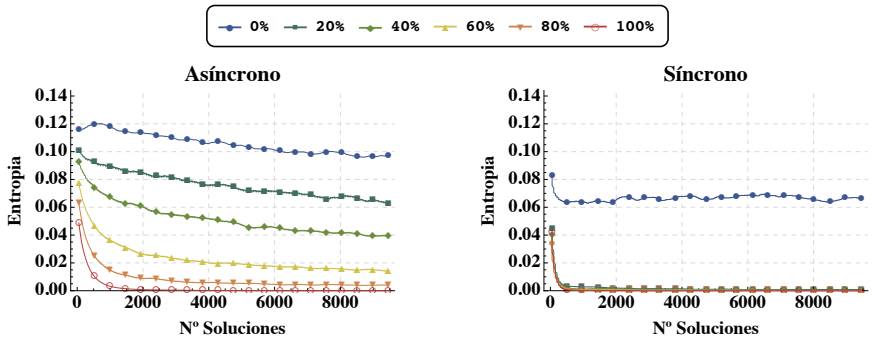


Figura 3.25: entropía media de la tabla de feromona empleando distintos porcentajes de foragers, utilizando la instancia *eil51*.

actualizaciones, los cambios de la primera actualización se van perdiendo debido a las actualizaciones posteriores.

Supongamos un recorrido de n ciudades, para la ciudad i la tabla indica que hay que visitar la ciudad j. Obviamente los foragers cuando actualicen la tabla para la ciudad i, reforzarán el visitar la ciudad j. En cambio, los patrollers pueden ignorar la tabla en algún punto de la construcción de una secuencia. Pero para cada ciudad en concreto, la probabilidad de usar la tabla por parte de un patroller es alta. Esto tiene como consecuencia que para la ciudad i, la mayoría de los patrollers también reforzarán la ciudad j, en cambio, unos pocos reforzarán otra opción:$i \rightarrow j'$. Si estos pocos actualizan la tabla en último lugar, es posible que la información quede reflejada en la tabla, pero justamente por ser un grupo escaso, lo más probable es que no sean los últimos en actualizar la tabla. La consecuencia es que ese cambio se pierde, en favor de la información antigua: $i \rightarrow j$.

Obviamente el uso de probabilidades, más la función de calidad que se asigna a una solución influye, permitiendo que la información se conserve mejor en función de la calidad de la solución que produce. Pero si la población es numerosa, aparece un efecto secundario debido a la repetición, que bien puede ser positivo: se refuerza una cierta información, o bien negativo: se decremento de manera repetitiva la probabilidad asociada a una acción.

En el esquema síncrono este efecto de repetición es muy significativo cuando la población alcanza un cierto número de individuos y la secuencia a construir es larga. Cuando la población es pequeña o la secuencia es corta, no se aprecia el efecto. En un esquema asíncrono este efecto no tiene una influencia apreciable, porque como cada hormiga actualiza la tabla en iteraciones distintas. Antes que posteriores actualizaciones eliminen cierta información de la tabla, existe la posibilidad de que dicha información se utilice por alguna hormiga que continúe buscando. La hormiga que la usa, la almacena en su memoria interna, de tal modo que cuando finalice su búsqueda, devolverá dicha información a la tabla.

Es decir, la información que determina el recorrido que una hormiga construye no se encuentra únicamente en la tabla, sino en las hormigas que llevan la construcción de solución más avanzada y que actualizarán la tabla mientras ella continúa buscando. En cambio, en el esquema síncrono esta información se encuentra únicamente de la tabla. Es la misma idea que se expuso de manera teórica al comparar el modelo biológico de las hormigas y de las abejas, al distribuir la

información entre los individuos se incrementa la información que maneja el algoritmo, lo que indirectamente se traduce en un mayor nivel de información en la tabla, como se aprecia en la figura 3.25.

Esta diferencia entre asíncrono y síncrono se ilustra mejor utilizando una instancia pequeña, 16 ciudades, y estudiando como evoluciona el contenido de la tabla durante la ejecución de las primeras 100 iteraciones.

Las figuras 3.26 y 3.27 muestran un "diagrama de temperatura". Para cada ciudad se muestra el contenido de la tabla (vector en la dirección del eje x) según la iteración (eje y). El código de colores varían entre más peso ($rojo = 1$) y menos peso ($blanco = 0$). Para esta pequeña simulación se ha utilizado una población de 16 hormigas con 7 foragers (40 %) y 9 patrollers (60 %).

Como se puede observar en el caso síncrono, la tabla se actualiza cada 16 iteraciones que es el periodo que tarda una hormiga en construir un recorrido. Para casi todas las ciudades se puede ver cómo la tabla muestra un patrón muy estable.

Por el contrario, la tabla en el caso asíncrono es mucho más dinámica, porque en cada iteración existe la posibilidad de que una hormiga la actualice si ha tenido éxito en la búsqueda. Aunque también se estabiliza, muestra una mayor diversidad de patrones que en el caso síncrono. Es interesante ver cómo para la ciudad 16 se produce una pequeña oscilación entre dos valores, indicativo de una convergencia a dos ciudades (acciones) distintas.

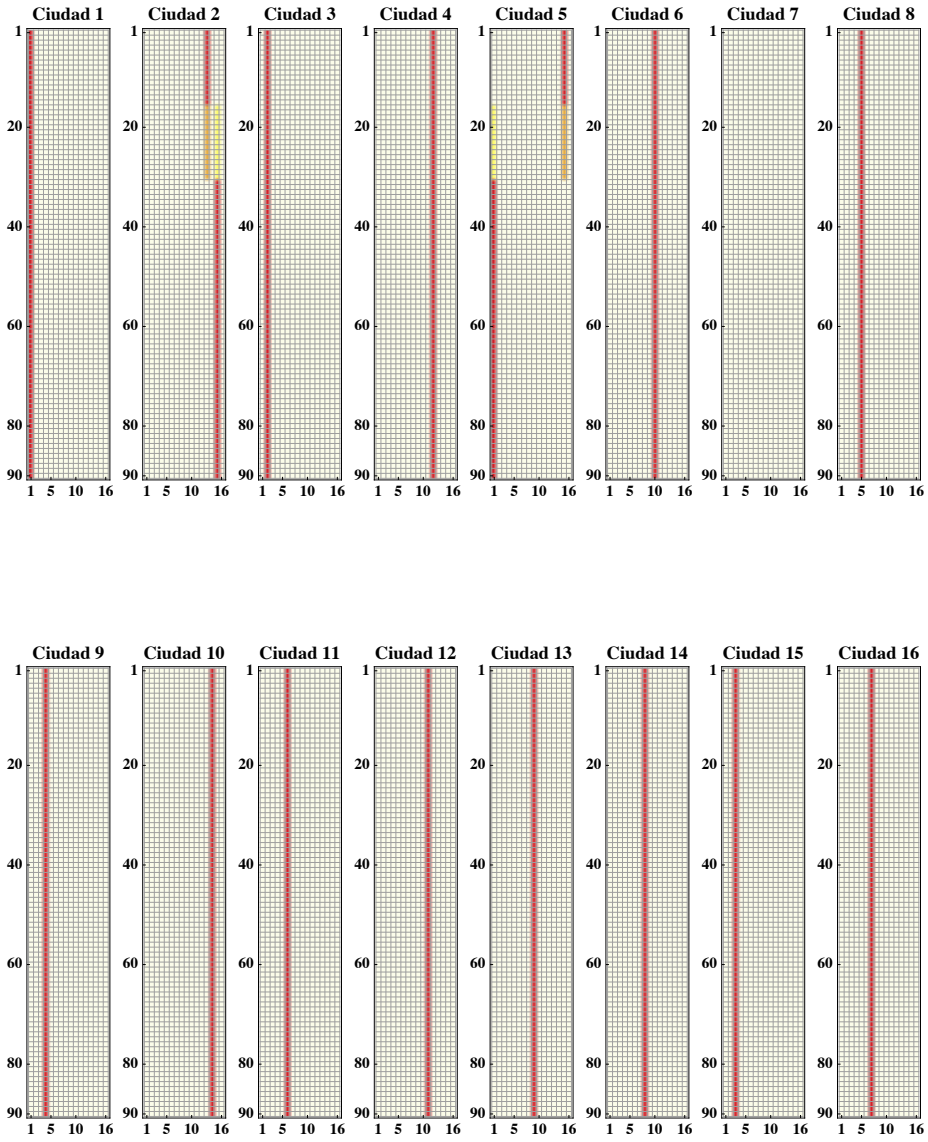


Figura 3.26: Evolución del contenido de la tabla de feromona para la versión síncrona, utilizando la instancia *ulysses16*.

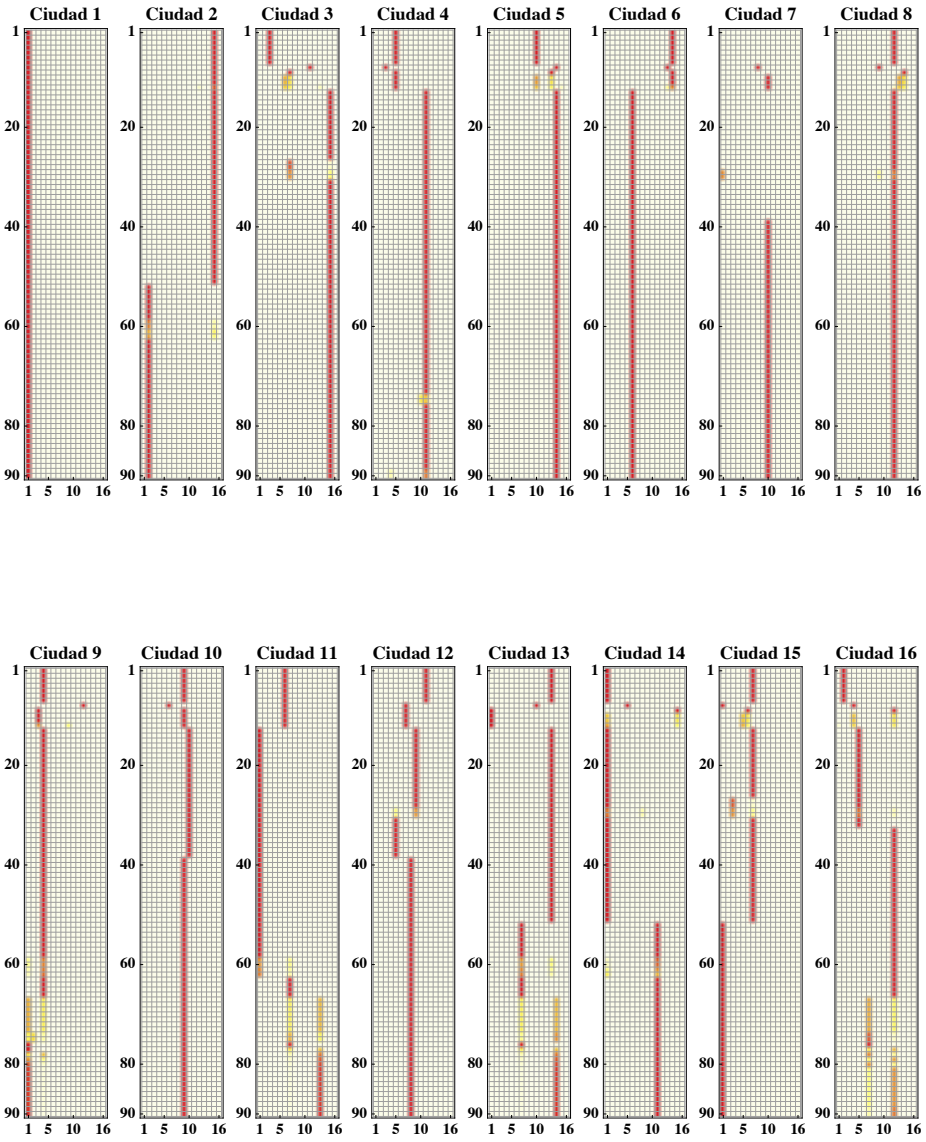


Figura 3.27: Evolución del contenido de la tabla de feromona para la versión asíncrona, utilizando la instancia *ulysses16*.

Soluciones

El análisis de la tabla es indicativo de como está realizando la búsqueda el algoritmo, pero para tener una medida más fiable es necesario analizar el conjunto de soluciones procesadas. Para realizar dicho análisis podemos recurrir de nuevo a la entropía, solo que aplicada al conjunto de recorridos construidos por las hormigas.

A medida que las hormigas construyen recorridos, los almacenamos en un conjunto y vamos midiendo la entropía de dicho conjunto. De esta manera obtenemos la evolución de la entropía del conjunto de soluciones generadas. Si los recorridos son diferentes, la entropía del conjunto debe incrementarse a medida que vamos añadiendo nuevas soluciones, en cambio, si las soluciones comienzan a repetirse la entropía decrecerá.

Al igual que en el caso anterior, el valor de la entropía se promedia utilizando 100 ejecuciones independientes.

El resultado de este análisis puede verse en la figura 3.28, como era de esperar el esquema asíncrono tiende a manejar un conjunto mayor de soluciones a lo largo de la búsqueda que el esquema síncrono. Únicamente cuando la población se compone exclusivamente de patrollers los conjuntos de soluciones presentan un nivel de entropía similar.

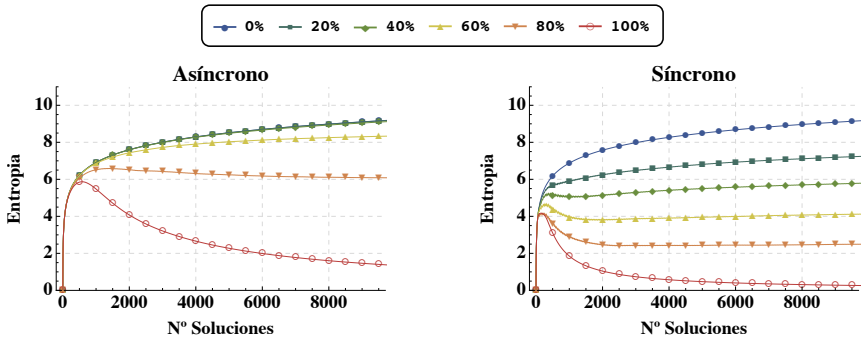


Figura 3.28: Entropía media del conjunto de soluciones, utilizando la instancia *eil51*.

Eficiencia

Por último, como estamos utilizando instancias conocidas del TSP, podemos medir la eficiencia real de cada esquema, utilizando el error relativo cometido respecto al óptimo:

$$\frac{coste - óptimo}{óptimo}$$

Para medir el error cometido, hemos seleccionado 10 instancias con diferente número de ciudades. Para cada instancia, hemos fijado la población de foragers a un 40 % con igual número de hormigas que ciudades hay en la instancia. El número de recorridos construidos para cada instancia es de $n \cdot 10^4$, donde n es el número de ciudades de la instancia. Cada instancia se resuelve 100 veces para obtener un valor promedio del error.

Los resultados se muestran en la figura 3.29, como puede verse el esquema asíncrono es claramente superior al esquema síncrono. Dicha diferencia se acentúa a medida que aumenta la complejidad de una estancia.

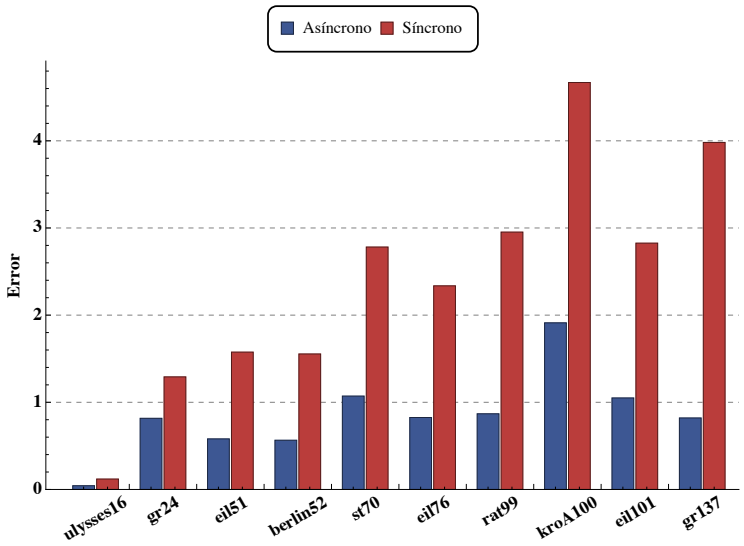


Figura 3.29: Error medio cometido al resolver cada instancia.

3.7. Recapitulación

Se adoptan ideas de la toma de decisiones colectivas de las colmenas de abejas, lo que permite ampliar la fuente de inspiración biológica de los algoritmos de hormigas. Esta inspiración se resume en dos ideas clave: a) los agentes trabajan de manera asíncrona, y b) utilizar la tabla como un buffer temporal.

El resultado es un sistema donde la información se distribuye entre los agentes, en vez de residir totalmente en la tabla. Esto permite una mayor eficacia en la búsqueda ya que la información contenida en los agentes permanece independiente una de otra, favoreciendo una dispersión global a nivel de sistema, pero permitiendo una convergencia local, a nivel de cada agente.

Los análisis experimentales realizados con un prototipo sugieren que la implementación de estas ideas permite aumentar la eficiencia de la búsqueda ya que se procesa una mayor cantidad de información y por tanto se consigue una mejor exploración del espacio de soluciones del problema.

Capítulo 4

Dinámica de población

En el capítulo anterior hemos introducido el esquema del prototipo, donde su característica principal es que la información se distribuye entre la población de agentes. Por tanto el rendimiento de la búsqueda depende en gran medida de la organización de esta población.

Como la información se distribuye entre los agentes, una población mayor puede contener más información y favorecer la dispersión en la búsqueda. Por el contrario, una población menor favorece la convergencia. También es importante considerar la composición de la población: un número elevado de patrollers induce una excesiva dispersión en la búsqueda y un número elevado de foragers, puede llevar al estancamiento de la búsqueda. Regular la población de manera adecuada se antoja como uno de los aspectos claves para pasar de un prototipo a un algoritmo que funcione de manera eficaz.

4.1. Alternativa al uso de parámetros

El esquema típico que se puede encontrar en muchos algoritmos meta-heurísticos al control[1] del proceso de búsqueda es la introducción de parámetros. Los procedimientos de generación de soluciones se regulan mediante constantes numéricas introducidas previamente a la ejecución del algoritmo, en función de dichos valores se controla el proceso de búsqueda. El uso de parámetros presenta dos dificultades:

a) Obtención de los valores adecuados. Previamente a la aplicación del algoritmo se debe buscar qué valores de los parámetros son adecuados para el uso del algoritmo.

   En muchos casos este proceso puede automatizarse, por ejemplo el algoritmo de optimización por descenso de gradiente [Sny05] consiste en avanzar en la dirección indicada por el gradiente. Ahora hay que definir "cuanto" se avanza en dicha dirección, para lo cual se puede definir dicho valor como una constante o utilizar un método de búsqueda lineal como por ejemplo el método de la bisección [Sny05].

[1] El control en este contexto consiste en definir un balance en el proceso de búsqueda: la generación de nuevas soluciones puede orientarse hacia la exploración de nuevas soluciones o la explotación de aquellos patrones de solución ya conocidos.

b) Excesiva rigidez. Al ser los parámetros valores predeterminados se pierde flexibilidad en el control del proceso. Por ejemplo, durante la ejecución del algoritmo pueden darse diferentes situaciones, donde cada una podría beneficiarse de un valor diferente de los parámetros de dicho algoritmo o de pequeños ajustes de dichos valores. Si los parámetros son constantes no existe tal posibilidad.

La dificultades inherentes al uso de parámetros pueden ilustrarse muy bien con el método de descenso por gradiente. El cual se basa en la siguiente ecuación:

$$x_{n+1} = x_n + \gamma \cdot \nabla F(x_n)$$

El valor de x_{n+1} se obtiene avanzando desde el valor anterior, x_n, en la dirección dada por el gradiente $\nabla F(x_n)$. La magnitud del avance viene controlado por el parámetro γ.

Si queremos aplicar el método del gradiente, la opción más simple es fijar γ a un valor razonable que determinamos a priori. Dicho valor puede funcionar muy bien durante casi todo el proceso de búsqueda excepto cuando el método se aproxima a un mínimo, donde comienza a oscilar debido a que el valor escogido de γ es demasiado grande. Podemos probar con otro más pequeño, pero eso ralentiza el tiempo de convergencia del método.

Una solución estándar es el uso de métodos de búsqueda lineales. Como tenemos fijos el valor de x_n y el valor del gradiente $\nabla F(x_n)$ aplicamos un método de búsqueda en un intervalo para determinar el valor de γ que mejor resultado ofrece.

Como se puede intuir, para cada paso de búsqueda del método del gradiente hay que ejecutar un segundo proceso de búsqueda que determine el valor de γ, incrementando el coste computacional del método de manera significativa. Lo que se suele hacer es aproximar el valor de γ para garantizar que es relativamente aceptable sin llegar a sobrecargar la búsqueda general.

La problemática del gradiente es extensible a métodos meta-heurísticos más complejos donde existan parámetros que controlan la búsqueda: en el momento que se intenta afinar el control de los parámetros, se introducen técnicas auxiliares que incrementan el coste computacional del método general. En general, cuando se diseña un algoritmo meta-heurístico es deseable evitar lo más posible la introducción de parámetros.

Diseñar un algoritmo carente de parámetros es una tarea casi imposible, pero se puede tratar de reducir su número. Si entendemos un parámetro como una forma de control de un proceso, entonces es posible sustituir dicho parámetro por otro tipo de estructura que cumpla el mismo rol. Una forma autónoma de controlar un proceso son las denominadas dinámicas auto-organizativas que consisten en la regulación mediante lazos de realimentación.

La organización de un sistema de agentes utilizando dinámicas auto-organizativas se enmarca dentro de dos enfoques clásicos de la Inteligencia Artificial respecto a la organización de un sistema.

Organización: información vs dinámica

Herbert Simon en su libro "The sciences of the Artificial" [Sim69] al abordar los sistemas complejos indica que para entender dichos sistemas hay que prestar especial atención a la jerarquía. En palabras de Simon un sistema jerárquico

es aquel que está compuesto por otros subsistemas interrelacionados entre ellos. Advierte Simon que el uso cotidiano de jerarquía lleva implícito una noción de subordinación, en el sentido que una parte se subordina a otra superior, pero que cuando se habla de sistema jerárquico no tiene porque existir dicha subordinación. Un sistema jerárquico lo que exhibe es una *gradación*, donde las partes más simples del sistema se agrupan en subsistemas más refinados.

En su libro "La sociedad de la mente" [Min88] M. Minsky también habla de agrupaciones de procesos, uno de sus ejemplos está representado en la figura 4.1. Como se puede observar los agentes se disponen de forma piramidal, de tal modo que únicamente los agente del nivel más inferior son capaces de interactuar con el mundo. En este ejemplo podemos ver diferentes subsistemas formados por 3 agentes.

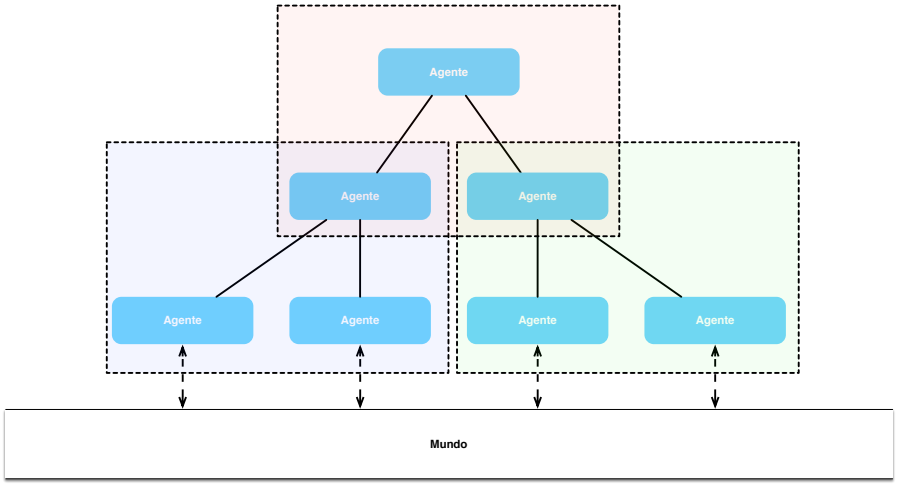


Figura 4.1: Estructura de una sociedad de agentes, tomada del libro "La sociedad de la mente" de M. Minsky. Los distintos recuadros son ejemplos de posibles agrupaciones entre los agentes.

Como se puede observar en la figura 4.1, cada agente del nivel más inferior interactúa con el mundo y por tanto es capaz de percibir cierta información. Dicha información será local, en el sentido que se localiza en el entorno del agente: hace referencia única y exclusivamente a aquella parte del mundo que el agente es capaz de percibir. La información global, referente al sistema completo, será la totalidad de las informaciones parciales de cada agente.

Si tuviésemos que describir el funcionamiento del sistema de la figura 4.1, podríamos simplemente decir que es una estructura jerárquica donde los agentes se relacionan entre ellos a través de información: cada agente de un nivel superior procesa la información proveniente de niveles inferiores, así sucesivamente hasta llegar a la cúspide de la pirámide. De este modo, el nivel superior tiene –o podría tener– acceso a toda la información del sistema. Una vez que la información se ha procesado podrían descender, desde los niveles superiores a los inferiores, las ordenes de actuación que deben realizar los agentes que están en contacto con el mundo.

Otro tipo de organización la podemos encontrar en una serie de artículos escri-

tos por Rodney Brooks [Bro90, Bro91] recogidos en el libro "Cambrian intelligence: the early history of the new AI" [Bro99]. En esta serie de artículos el autor propone un cambio de perspectiva respecto al diseño de los sistemas tradicionales en Inteligencia Artificial:

> The fundamental decomposition of the intelligent system is not into independent information processing units which must interface with each other via representation. Instead, the intelligent system is decomposed into independent and parallel activity producers which interface directly to the world through perception and action.

En la figura 4.1 cada subsistema es una unidad de procesamiento de la información y la relación entre los subsistemas es información que circula. Brooks, en cierta medida, sugiere que el mundo ya es en sí un canal de información. De tal modo que un agente o un elemento de un sistema actúa sobre el mundo. Al actuar cambia el estado del mundo y dicho cambio puede ser percibido por otro componente del sistema que actúa en consecuencia. Un ejemplo de este tipo de organización se ilustra en la figura 4.2, como se puede observar no existe ninguna via de transmisión de información entre los agentes, sino que la influencia de un agente a otro es a través de una acción.

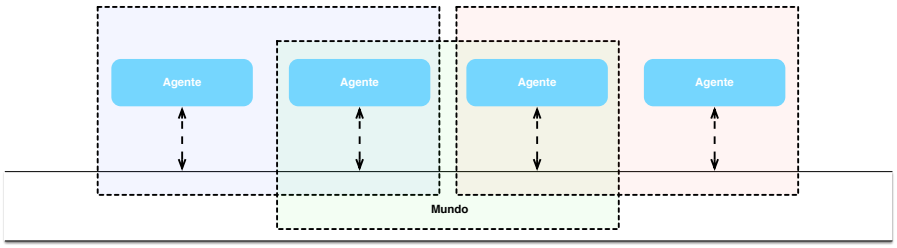


Figura 4.2: Estructura de un sistema de agentes relacionados a través del mundo. Los distintos recuadros son ejemplos de posibles agrupaciones que pueden darse entre los agentes.

Desde el punto de vista de la organización, podemos decir que el sistema de la figura 4.1 se organiza en función de la información: el sistema procesa información y dicho procesamiento determina la actuación de los agentes y por ende el comportamiento del sistema. Ahora bien, para que un sistema utilice información es necesario definir las vías de transmisión. En tanto que dichas vías establecen relaciones entre los elementos o agentes del sistema y son definidas de antemano, la estructura u organización del sistema está definida de manera previa.

El sistema de la figura 4.2 se organiza de manera dinámica. Cada agente puede realizar una serie de acciones, la cuales determinarán el comportamiento del resto: reforzando o inhibiendo ciertas conductas en función de las estructuras de realimentación presentes. No existe ninguna organización previa, pues únicamente al actuar es cuando los elementos se relacionan entre sí. Nótese que si se elimina un agente del sistema este puede continuar funcionando, de otra manera, pero es posible su funcionamiento. Por el contrario, en el sistema basado en la información, si eliminamos un agente, el procesamiento ya no será completo y el sistema puede no funcionar.

La organización mediante información permite el desarrollo de conductas planificadas, lo que conlleva un desempeño eficaz de un determinado comportamiento. Ahora bien, para llevar a cabo un determinado planificación es necesario que se den unas determinadas circunstancias, si éstas no se dan, el plan no se puede llevar a cabo y el sistema no será operativo.

En cambio, una organización dinámica es puramente reactiva, su propósito también puede ser el desarrollo de una cierta conducta. No importa tanto si dicha conducta se realiza de forma eficaz o no, sino que se realice. Es decir, el objetivo final es la realización de un cierto comportamiento "como se pueda" sin importar las circunstancias.

Optar por una organización dinámica, permite el diseño de un algoritmo reactivo: él mismo es capaz de organizarse en función del problema que le sea dado. Es claro que se pierde control, no es posible un ajuste tan fino como permite una parametrización y teóricamente se pierde rendimiento. Ahora bien, dicho ajuste fino hay que encontrarlo, lo cual compensa si se está desarrollando un algoritmo o método ad-hoc para un cierto problema. Si lo que se pretende es un algoritmo de propósito general, que se va a aplicar a una colección de problemas amplia, el enfoque dinámico es más flexible.

4.2. Auto-organización en sistemas biológicos

> Self-organization is a process in which pattern at the global level of a system emerges solely from numerous interactions among the lower-level components of the system.
>
> S. Camazine et al. "Self-Organization in Biological Systems"

Como indica la cita, la auto-organización está ligada a la aparición de patrones, los cuales emergen de las interacción entre los elementos de un sistema. Para analizar cómo interactúan los elementos de un sistema, vamos a introducir brevemente la denominada dinámica de sistemas.

La dinámica de sistemas consiste en el estudio de la evolución del sistema. Como realmente cualquier objeto puede ser considerado un sistema, la dinámica de sistemas se puede aplicar en campos muy diversos, véase por ejemplo [Gha11, Mea08]. En nuestro caso nos interesan las técnicas de representación (diagramas causales) porque permiten obtener un esquema formal de los procesos de auto-organización de un sistema biológico. Al ser un esquema formal, se puede extraer del contexto biológico y trasladarlo a un contexto computacional.

Las colonias de insectos constituyen lo que se conoce como "super-organismos", donde los individuos se organizan en una sociedad [HW90, HW09, Wil00]. Esto significa que los insectos actúan de manera conjunta para llevar a cabo ciertas tareas. Un insecto es un organismo muy simple, carente de aprendizaje o inteligencia individual, todo el comportamiento complejo que exhibe una colonia es debido a la interacción que se da entre sus individuos.

Deborah Gordon: organización en una colonia de hormigas

Los estudios biológicos de Deborah Gordon [Gor10, Gor99, Gor91, Gor89, GK96, Gor96] revelan que una colonia de hormigas auto-regula su organización a través de las interacciones –encounters– entre las hormigas, es decir, comunicación directa. Esta organización consiste principalmente en la forma de distribuir

las hormigas entre las diferentes tareas que deben realizarse. La organización está basada en los roles –patrones de comportamiento–, donde cada rol está asociado a la realización de una tarea específica[2].

Antes de entrar en los detalles del trabajo de Gordon conviene introducir varios elementos clave:

Distribución de tareas: los comportamientos complejos de una colonia surgen a medida que las hormigas realizan diferentes tareas sencillas. Es común diferenciar entre *hormigas activas*, aquellas que ya están desempeñando una tarea, y *hormigas inactivas*, aquellas que aún no están comprometidas con ninguna tarea.

Reclutamiento: una hormiga que esté realizando una cierta tarea, puede estimular a otra hormiga que se encuentre inactiva para que comience a desempeñar la misma tarea que ella realiza, u otra distinta.

Activación espontánea: un mecanismo que permite a las hormigas inactivas comenzar a desempeñar una tarea sin necesidad de ser estimuladas por otra hormiga activa.

La organización de las hormigas descrita por Gordon se basa en el concepto de *encounter*, el cual no es más que el *encuentro* de dos hormigas. El resultado de dicho encuentro viene determinado por las tareas que realiza cada hormiga, dando lugar a un cambio de roles (tareas). Por ejemplo, una hormiga está extrayendo residuos del interior del hormiguero, se encuentra con una hormiga inactiva y de dicha interacción resulta que la hormiga inactiva comienza a su vez la extracción de residuos.

Anteriormente hemos hablado de la diferencia entre intercambio de información y actividad. El encuentro entre dos hormigas no suponen ningún intercambio de información, simplemente una hormiga está llevando a cabo una actividad –realizando una acción– y se encuentra con otra. Al encontrarse dos actividades distintas[3], se produce una "influencia" mutua, dando lugar a un cambio y permitiendo la organización de los individuos a través de la acción, sin necesidad de intercambiar información. Por ejemplo, una hormiga no le indica a otra si hay muchos residuos que extraer o pocos, esa información no es necesaria para la organización de la colonia.

Como se puede intuir, los *encuentros* forman un mecanismo de reclutamiento mediante lazos de realimentación positivos. Cuantas más hormigas realicen una determinada tarea, más encuentros habrá donde participe una hormiga que realiza dicha tarea, por tanto más hormigas se verán asignadas a dicha tarea. Es decir, el desempeño de una tarea tiene como consecuencia un reclutamiento de nuevas hormigas a dicha tarea.

Por último, es necesaria una activación espontánea, puesto que si todas las hormigas se encuentran inactivas, no existe posibilidad de reclutamiento.

Veamos estos conceptos dentro de una actividad determinada: la recolección de comida. Dicha actividad se compone de dos tareas: la localización de las fuentes

[2] Aunque, todas las hormigas son idénticas, es común diferenciar distintos tipos de hormigas de acuerdo con los diferentes roles.

[3] La inactividad también es una actividad, especial porque consiste en "no hacer nada", pero no deja de ser un patrón de comportamiento.

de alimento y la recogida de dicho alimento. Por tanto se necesitan dos tipos de hormigas[4] (dos tareas):

Patrollers: encargadas de la localización de las fuentes de comida.

Foragers: encargadas de la recolección de alimento.

La recogida de alimento funciona de la siguiente manera:

I. Comienza con la activación espontánea de los patrollers.

II. Un patroller que logra localizar una fuente de alimento regresa al hormiguero, *encontrándose* con hormigas inactivas. El resultado de este encuentro es un reclutamiento de hormigas inactivas, las cuales pasan a adquirir el rol de foragers. En otras palabras, los patrollers que han tenido éxito reclutan foragers para que dé comienzo la tarea de recolección de alimento.

III. Una vez que un pequeño grupo de foragers han sido activados, la actividad de recolección es auto-sostenida, ya que los foragers también pueden reclutar nuevos foragers:

- Cuando un forager tiene éxito en la explotación de una fuente de alimento, vuelve al hormiguero portando alimento. Cuando esto ocurre, dicho forager estimula a hormigas inactivas para que a su vez realicen la tarea de recolección de alimento, es decir, recluta nuevos foragers.
- En el caso de que un forager no encuentre comida, vuelve al hormiguero y se torna inactivo, abandonando la tarea de recolección.

La descripción dada por Gordon la podemos representar utilizando un diagrama causal para ilustrar cuál es la dinámica del sistema. El diagrama se representa en la figura 4.3, como se puede observar, hay dos lazos de realimentación que regulan el número de foragers activos. El de la derecha es un bucle de realimentación positiva: mientras más foragers estén activos, más foragers volverán al hormiguero llevando comida, reclutando otros nuevos y aumentando el número de foragers activos. El de la izquierda es un bucle de realimentación negativa: los foragers que vuelven al hormiguero sin alimento pasan a ser hormigas inactivas, con lo cual el número de foragers activos disminuye. En la parte superior se puede observar la activación externa por parte de los patrollers.

Respuesta a un estímulo Gordon describe toda la actividad de recolección utilizando el concepto de *encuentro*, donde una hormiga puede estimular a otra para comenzar a realizar una cierta tarea. Pero esto sólo ocurre si el estímulo generado es mayor que un cierto umbral.

Por ejemplo, una hormiga adquiere el rol de forager únicamente si su tasa de encuentro con hormigas patroller es mayor que un cierto umbral. Es decir, los encuentros esporádicos, muy espaciados en el tiempo, no producen reclutamiento, ya que el efecto producido por un encuentro decae con el tiempo. Este comportamiento tiene sentido debido a que un alto porcentaje de encuentros significa que hay abundancia de alimentos disponibles y por lo tanto la actividad de recogida

[4] Los nombres dados a los diferentes agentes se tomaron de los nombres originales utilizados por Gordon

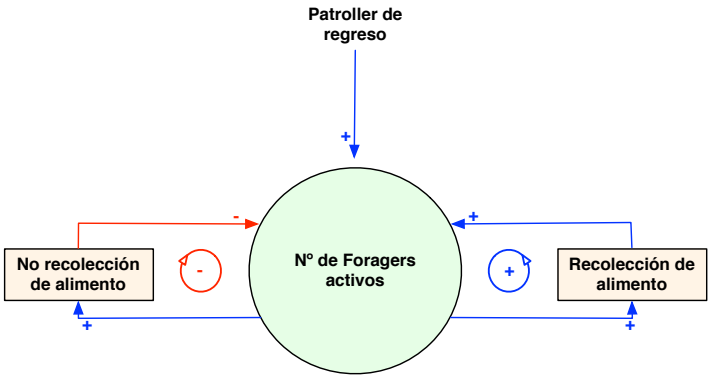


Figura 4.3: Diagrama causal de la actividad de recolección.

merece la pena. Por otro lado, una baja tasa significa que el alimento es escaso o demasiado disperso, y por lo tanto su recogida no merece la pena.

La respuesta a un estímulo utilizando un umbral es una forma común para modelar la división de tareas en los insectos sociales [BTD96, BDT99]. En estos modelos, es común que los estímulos sean generados por el medio, permaneciendo externos a los insectos. Por ejemplo, el alimento disponible que puede ser percibido. Además, los estímulos disminuyen a medida que se realiza la tarea.

Deborah Gordon propone que los estímulos se producen por la actuación de la tarea, no son externos. Es la actividad de recogida en sí misma la que produce los estímulos necesarios para mantener la propia actividad funcionando.

Por ejemplo, vamos a suponer una pequeña fuente de alimento cerca de la entrada de una colonia. La comida es escasa, pero ya que está cerca de la colonia, es fácil de encontrar. Si las hormigas reaccionan a la cantidad de alimento disponible, estímulos externos, podrían ignorarla ya que no es suficientemente abundante. Sin embargo, si los estímulos son producidos por la actividad de recogida, la colonia probablemente explote esta pequeña fuente de alimento, ya que sólo se necesita que unas pocas hormigas lleven algo de comida para estimular la actividad de recogida. Una vez que la actividad se inicia, no depende de la cantidad disponible de alimentos, sino de las posibilidades de encontrarlo: del desempeño de la propia actividad de recogida.

El comportamiento de las hormigas de Gordon es muy interesante ya que podemos definir una dinámica que organice la población de agentes del sistema en función de su propia actividad, sin recurrir a elementos externos. La propia búsqueda de los agentes determinará su organización.

Agentes de búsqueda y auto-organización

Utilizando las ideas de Gordon acerca de la recolección de comida en una colonia de hormigas, podemos definir un modelo de auto-organización similar pero en un entorno computacional.

En la figura 4.4 se puede ver el esquema básico de esta dinámica. Se trata del número de agentes asignados a cierta tarea, a medida que los agentes culminan la tarea con éxito, se vuelven inactivos y dejan de desempeñar la tarea. Si los

agentes fracasan en el desempeño de la tarea, se refuerza el número de agentes asignados (reclutamiento) para incrementar el esfuerzo destinado al cumplimiento de la tarea. La activación externa permite iniciar el proceso.

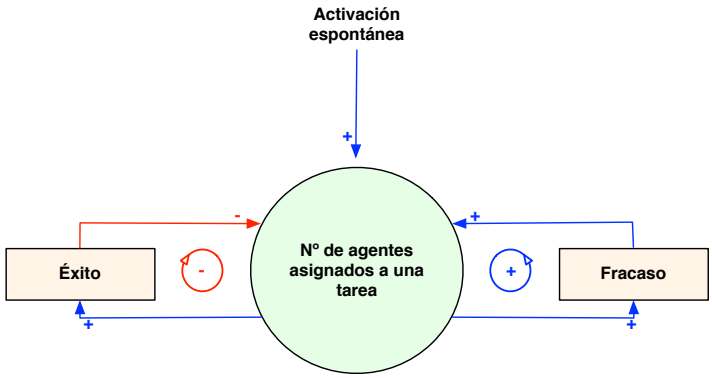


Figura 4.4: Diagrama causal de una dinámica básica para asignar agentes al cumplimiento de una tarea.

La idea de esta dinámica es sencilla: mientras sea difícil resolver una tarea, se incrementarán el número de agentes asignados a ella, a medida que se va completando la tarea, se reduce el número de agentes asignados a su consecución. De esta dinámica (figura 4.4) podemos extraer la idea principal que nos servirá para diseñar la auto-organización del sistema:

> *La auto-organización del sistema se consigue creando lazos de refuerzo positivo entre los agentes a través de sus lazos negativos. De este modo a medida que la actividad de un agente decae se estimula la actividad del resto.*

La auto-organización está basada en lazos de realimentación que fomentan o inhiben una determinada actividad (tarea). Una búsqueda no deja de ser una actividad, por tanto, la misma dinámica previamente introducida se puede particularizar para el caso de que la tarea consista en buscar. De este modo podemos controlar un sistema multi-agente de búsqueda mediante una dinámica auto-organizativa.

El esquema de la dinámica sería el que se muestra en la figura 4.5. El sistema dispone de una inicialización que genera una serie de patrollers. Estos primeros agentes son los encargados de realizar las primeras búsquedas. Los patrollers que tienen éxito en la búsqueda generarán nueva información. Inmediatamente estos agentes pasarán a ser inactivos y estimularán la generación de foragers, igualmente cuando estos agentes tengan éxito, se volverán inactivos y activarán el reclutamiento de patrollers.

Como disponemos de dos tipos de agentes que buscan de manera distinta, el balance de la población se traduce en el balance de la búsqueda[5]. El uso de una dinámica de auto-organización permite definir el balance de la búsqueda a través de un balance de la población:

[5]Por esta razón cuando diseñamos el prototipo introdujimos los dos tipos de agentes.

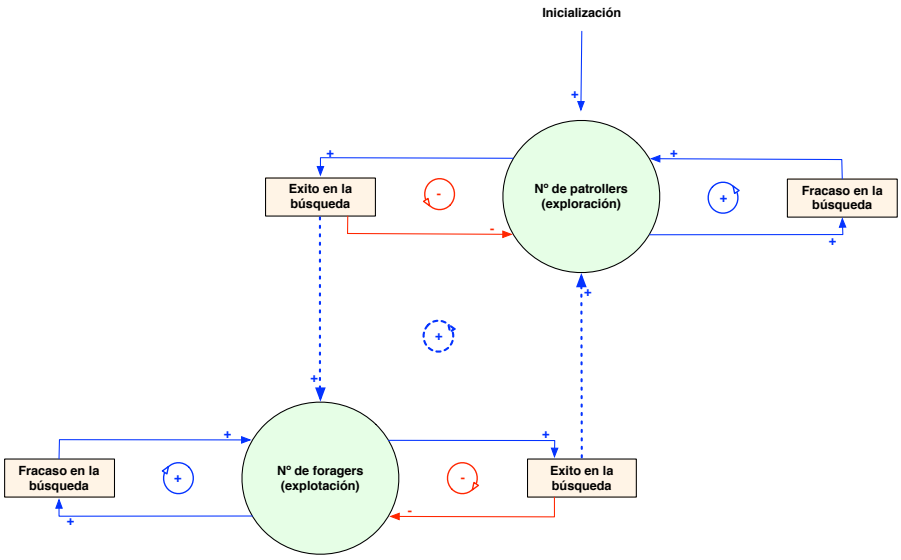


Figura 4.5: Diagrama de la dinámica auto-organizativa para controlar la población de agentes del sistema.

Exploración: cuando la población de patrollers domina sobre la de foragers, se produce una *ganancia de información* que permite incrementar la dispersión en la búsqueda.

Explotación: cuando la población de foragers domina sobre la de patrollers, se produce una *disminución de información* que permite reducir la dispersión en la búsqueda.

Es interesante advertir que la auto-organización permite el equilibrio entre ambos tipos de búsqueda gracias a los diferentes lazos de realimentación que se dan en el sistema:

- *Realimentación positiva local:* se garantiza la consecución de cada búsqueda (tarea), puesto que hasta que no se realiza con éxito los agentes no se vuelven inactivos.
- *Realimentación positiva cruzada:* cuando un agente se vuelve inactivo es que ha tenido éxito, por tanto incrementa el número de agentes asignados a otro tipo de búsqueda. De este modo se consigue equilibrar cada tipo de búsqueda para realizarlas de manera ponderada.

A su vez, la auto-organización permite introducir en el sistema capacidades *reactivas*, el sistema reacciona en función de lo que ocurra, adecuando la distribución de los agentes. La reacción siempre es en el sentido opuesto de lo que está ocurriendo: si la información aumenta, el sistema reacciona disminuyéndola, y viceversa: cuando la información disminuye, el sistema trata de incrementarla. De este modo el sistema trata de encontrar siempre un equilibrio, donde no haya excesiva información, pues dispersaría la búsqueda, ni lo contrario: escasez de información, lo cual estancaría la búsqueda.

Éxito en la búsqueda la auto-organización requiere un criterio que permita clasificar cuando la búsqueda (la tarea) se ha llevado a cabo satisfactoriamente o no. Este criterio ya está incluido en el prototipo: el mismo que sirve para determinar cuando se lleva a cabo una actualización de la tabla de feromona:

$$\begin{cases} J(s) \leq \mu & \rightarrow \acute{E}xito \\ J(s) > \mu & \rightarrow Fracaso \end{cases}$$

4.3. Implementación

En esta sección vamos describir la implementación de la dinámica de población basada en la ideas expuestas en la sección 4.2.

La dinámica de población se basa en el uso de dos señales:

- UP: número de patrollers sin éxito.

- SP: número de patrollers con éxito.

Las señales simulan estímulos, de tal modo que cuanto mayor es el valor de la señal, más intenso podemos considerar el estímulo. La intensidad de una señal determina la activación de un agente, de igual modo que la intensidad de un estímulo determina la activación de una hormiga.

Antes de describir la implementación de la dinámica de población, es importante advertir que poblaciones con un número elevado de foragers que conducirían al estancamiento de la búsqueda. Para evitar esto, vamos a recurrir al uso de funciones de respuesta a un umbral.

Estas funciones ya la hemos mencionado antes: sirven para modelar la respuesta de un individuo dependiendo de si la intensidad de un estímulo supera un cierto valor umbral [BTD96, BDT99].

Existen diferentes formas de definir una función de respuesta, en nuestro caso al ser un sistema discreto son simples funciones de escalón:

$$T_\theta(s) = \begin{cases} 1 & s \geq \theta \\ 0 & s < \theta \end{cases}$$

donde s es el valor del estímulo y θ es el umbral.

Podemos utilizar las funciones de umbral para definir un reclutamiento de foragers de acuerdo al valor de la señal SP pero teniendo en cuenta el número actual de foragers:

$$T_{FR}(sp) = \begin{cases} 1 & SP \geq FR \\ 0 & SP < FR \end{cases} \tag{4.1}$$

donde FR indica el número actual de foragers activos.

Esta función permite a los foragers reaccionar al valor de SP pero teniendo en cuenta su número actual, de tal modo que no se activan más foragers de los necesarios. De esta manera se evitan posibles situaciones de sobre-explotación que conducirían al estancamiento de la búsqueda.

Descripción

La dinámica de población se ilustra en el esquema mostrado en la figura 4.6. Existe una primera fase de inicialización, la cual no se muestra la figura, que juega el papel de la activación espontánea en la dinámicas biológicas: hasta que se encuentra una primera solución, se activa un patroller por ciclo. Esta primera solución es necesaria para disponer de un valor para el umbral y poder clasificar los éxitos y los fracasos.

Tampoco se muestra en la figura, pero cuando un agente finaliza una búsqueda, independientemente de si ha tenido éxito o no, el agente pasa a ser inactivo. Es la misma idea que un sistema biológico: los individuos tienden a la inactividad, a menos que exista un estímulo que los obligue a realizar una determinada tarea.

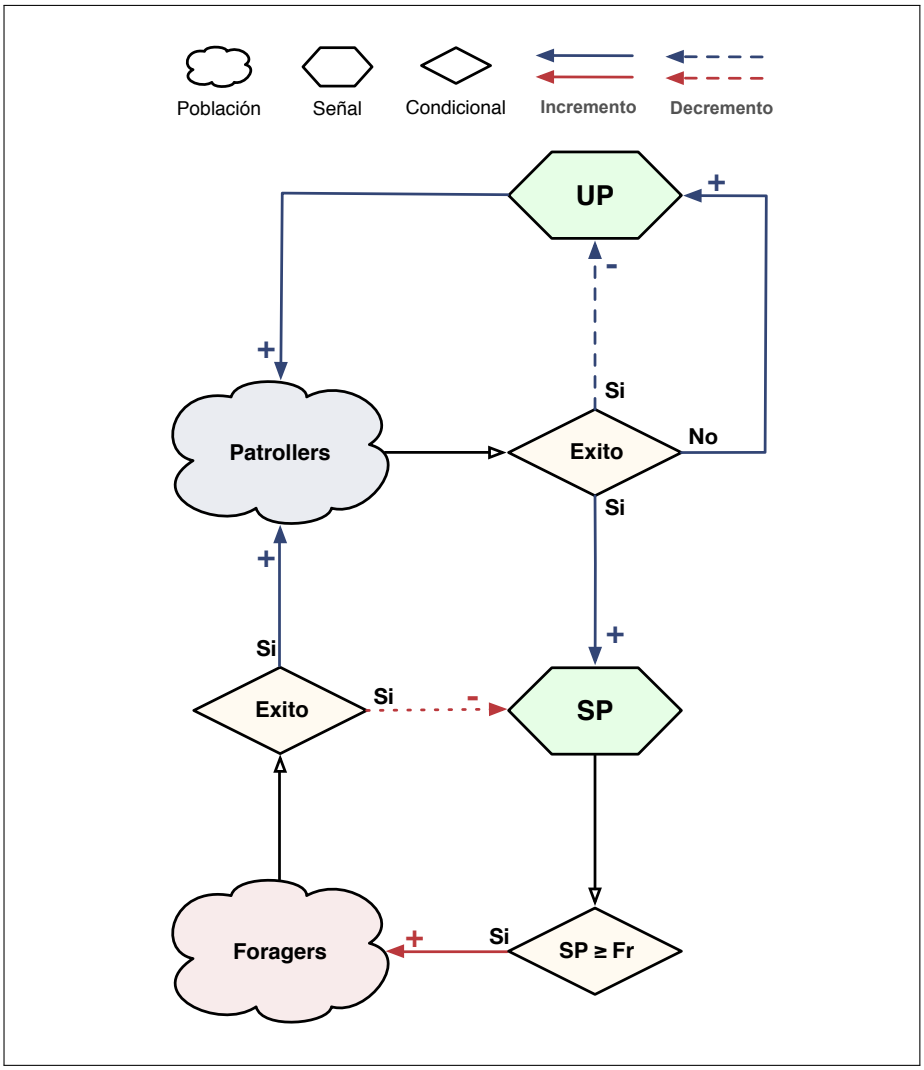


Figura 4.6: Esquema de la dinámica de población. UP = nº de patrollers sin éxito, y SP = nº de patrollers con éxito.

El comportamiento del sistema emerge de la reacción a los eventos que pueden ocurrir:

- Si un patroller fracasa, desencadena un evento que incrementa el valor de la señal UP.

  El incremento de la señal UP tiene como consecuencia el incremento de la población de patrollers. De este modo se implementa un reclutamiento de patrollers en base a la tasa de fallo de los propios patrollers. Utilizando un lazo de refuerzo positivo, se ajusta el esfuerzo que dedica el sistema (número de patrollers) en función de la dificultad de tener éxito en la búsqueda.

- Si un patroller tiene éxito, desencadena un evento que incrementa el valor de la señal SP y decrementa el valor de la señal UP.

  Cuando los patrollers tienen éxito, inhiben el valor de la señal UP, de este modo se detiene el reclutamiento y se hace posible la disminución de la población de patrollers. A su vez, se incrementa el valor de la señal SP, la cual activa a la población de foragers.

- Si un forager tiene éxito, desencadena un evento que decrementa el valor de la señal SP y estimula la activación de patrollers.

  Como se puede observar, no hay reclutamiento de foragers en caso de que fallen. La razón es que podemos asumir que la probabilidad de éxito de un forager es ≈ 1, y por tanto no necesitamos el lazo de refuerzo positivo.

El sistema se auto-organiza en función de la frecuencia de los eventos, lo que le permite adaptar su comportamiento de acuerdo a las diferentes situaciones que puedan ocurrir. Por ejemplo, si los patrollers no tienen éxito con frecuencia, el sistema tiende a reclutar nuevos patrollers. Al mismo tiempo, el número de foragers disminuye, ya que estos terminan sus búsquedas y no existen suficientes patrollers con éxito para estimular la activación de nuevos foragers, desplazando el carácter de la búsqueda hacia una exploración.

Implementación de la dinámica de población

Los siguientes procedimientos implementan la regulación de la población de agentes del algoritmo. Existen tres procedimientos: $\acute{e}xito$, $fracaso$, y $reclutamiento$. Los cuales comparten las siguientes variables (contadores):

- RP, número de patrollers a reclutar.
- RF, número de foragers a reclutar.
- UP, número de patrollers sin éxito desde el último patroller con éxito.
- SP, número de patrollers con éxito desde el último forager con éxito.
- FR, número de foragers activo.
- PR, número de patrollers activo.

Éxito

La figura 7.5 muestra el pseudo-código del procedimiento $Exito(ant)$:

- Un forager con éxito recluta un patroller y pone a cero el contador SP, inhibiendo de este modo el reclutamiento de nuevos foragers.
- Un patroller con éxito incrementa el valor del contador SP y recluta un forager dependiendo del valor de SP y del número de foragers activos. En caso de no producirse un reclutamiento de un forager, se recluta un patroller para mantener el tamaño de la población.

```
1:  procedure Éxito(ant)
2:      if ant es Forager then
3:          FR ← FR − 1
4:          SP ← 0
5:          RP ← 1
6:      else
7:          PR ← PR − 1
8:          SP ← SP + 1
9:          UP ← 0
10:         if SP ≥ FR then
11:             RF ← 1
12:         else
13:             RP ← 1
14:         end if
15:     end if
16: end procedure
```

Figura 4.7: Descripción en pseudo-código del procedimiento $Exito(ant)$.

Fracaso

La figura 7.6 muestra el pseudo-código del procedimiento $Fracaso(ant)$:

- Cuando un forager fracasa se aplica el siguiente criterio para determinar si fuese necesario reclutar un nuevo forager o no. En caso de no serlo, simplemente se recluta un patroller para mantener el tamaño de la población.

$$RF^{+} = \begin{cases} 1 & \left(1 - \frac{J_{best}}{\mu}\right) > \frac{FR}{FR+PR} \\ RF & \text{e.o.c} \end{cases} \tag{4.2}$$

$$RP^{+} = \begin{cases} 1 & \left(1 - \frac{J_{best}}{\mu}\right) \leq \frac{FR}{FR+PR} \\ RP & \text{e.o.c} \end{cases} \tag{4.3}$$

La diferencia entre el valor umbral, μ, y el coste de la mejor solución descubierta J_{best} sirve como estimación de la probabilidad de que un patroller

tenga éxito. Si dicha diferencia es superior a la proporción de foragers en la población ($\frac{FR}{FR+PR}$), entonces se activa un forager para ajustar el balance de la población.

- Un patroller que fracasa incrementa el valor de la señal UP y activa el reclutamiento de nuevos patrollers de acuerdo al logaritmo del valor de dicha señal: $round(log(UP+1))$. De este modo, a medida que los patrollers fracasan se van activando nuevos teniendo en cuenta el número de fracasos anteriores.

  Como la generación de patroller según su tasa de fallos es un realimentación positiva, puede darse el caso de que la población crezca rápidamente. Si la población de foragers no aumenta de manera proporcional, y un cierto número de patrollers tienen éxito, entonces se podría generar una excesiva dispersión en la búsqueda. Para evitar este problema, cuando reclutamos patrollers, comprobamos si es necesario reclutar un forager según el criterio de la ecuación 4.2, para mantener un equilibrio mínimo entre las poblaciones.

```
1:  procedure FRACASO(ant)
2:      if ant es Forager then
3:          FR ← FR − 1
4:          if (1 − J_best/μ) > FR/(FR+PR) then
5:              RF ← 1
6:          else
7:              RP ← 1
8:          end if
9:      else
10:         PR ← PR − 1
11:         UP ← UP + 1
12:         RP ← round(log(UP + 1))
13:         if (1 − J_best/μ) > FR/(FR+PR) then
14:             RF ← 1
15:         end if
16:     end if
17: end procedure
```

Figura 4.8: Descripción en pseudo-código del procedimiento $Fracaso(ant)$.

Reclutamiento

Figura 7.7 representa el procedimiento de reclutamiento. Hasta que se descubre una primera solución ($muestras < 1$), se activa un patroller cada ciclo. Esto constituye la fase de inicialización. Una vez que se dispone de esa primera solución, se opera de manera normal de acuerdo a los valores de RP y RF.

```
1: procedure RECLUTAMIENTO
2:     if muestras < 1 then                          ▷ Fase de inicialización
3:         Población ← activarPatroller()
4:         PR ← PR + 1
5:     else
6:         while RP > 0 do
7:             Población ← activarPatroller()
8:             PR ← PR + 1
9:             RP ← RP − 1
10:        end while
11:        while RF > 0 do
12:            Población ← activarForager()
13:            FR ← FR + 1
14:            RF ← RF − 1
15:        end while
16:    end if
17: end procedure
```

Figura 4.9: Descripción en pseudo-código del procedimiento *Reclutamiento*().

Simulación

Para entender el funcionamiento de la dinámica y su lógica vamos a recurrir a unas sencillas simulaciones. El esquema de simulación es el siguiente:

- Un agente tarda 10 iteraciones en construir una secuencia.
- Definimos una probabilidad de éxito para la búsqueda de un agente. En las más sencillas asumiremos que esta probabilidad es 1 y después introduciremos valores más reales.
- Como las señales UP y SP son contadores podemos alterar su valor en cada iteración para simular distintas condiciones. Por ejemplo, aumentamos en una unidad la señal UP por cada iteración, simulando que un patroller fracasa en cada iteración. De este modo podemos ilustrar las poblaciones por separado en condiciones sencillas.

El propósito de estas simulaciones es ilustrar como se relaciona de manera independiente cada población con la señal que gobierna el numero de agentes. Después se vincular el efecto de una población sobre la señal de gobierno de la otra población, para mostrar como es posible sincronizar ambas poblaciones. Finalmente, simulamos un caso realista para mostrar un el comportamiento final que se obtiene.

Simulación 1: evaluación independiente.

En esta primera simulación vamos a evaluar el reclutamiento de foragers y patrollers de manera independiente. Lo vamos a hacer desacoplando las señales y evaluando por separado como afecta la señal SP a la población de foragers y como la señal UP afecta a los patrollers.

Para esta simulación establecemos lo siguiente:

- Un agente siempre tiene éxito:

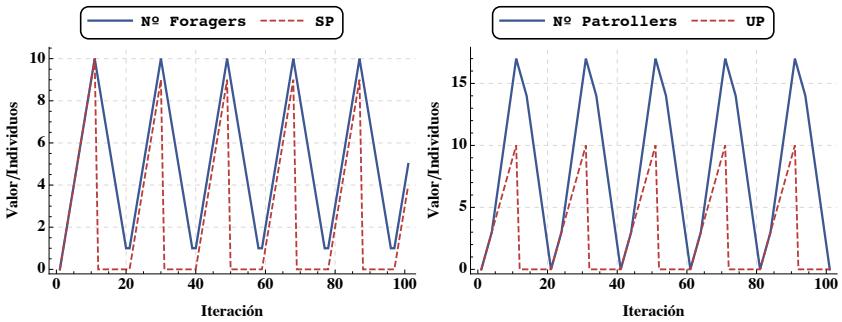
$$P(\text{Éxito}|Forager) = 1$$
$$P(\text{Éxito}|Patroller) = 1$$

- Simulamos los incrementos en la señales que manejan las poblaciones:

$$UP[i+1] = UP[i] + 1$$
$$SP[i+1] = SP[i] + 1$$

Los resultados de la simulación se muestran en la figura 4.10. Para el caso de los foragers, la idea es generar foragers en función del éxito de los patrollers, el cual se contabiliza en la señal SP, teniendo en cuenta los foragers ya activos para evitar reclutar foragers en exceso. La población se controla mediante la condición: $SP > FR$, a medida que se incrementa el valor de la señal se activa un nuevo forager hasta la iteración 10. En dicha iteración, el primer forager activado ha terminado de construir una secuencia y pone el valor de $SP = 0$. Según van finalizado los foragers, ponen dicha señal a cero, lo que resulta en una inhibición en la activación de los foragers durante 10 iteraciones puesto que $SP < FR$. Finalmente comienza el reclutamiento otra vez.

Para el caso de la población de patroller, estos se activan según el logaritmo del valor de UP. Al igual que los foragers, a medida que los patrollers finalizan su búsqueda se inhibe la activación de nuevos patrollers manteniendo el valor de la señal UP a cero. En un caso real, la señal UP sirve como histórico de los fallos acumulados. Cuanto mayor sea el número de fallos, más difícil es tener éxito en la búsqueda, por tanto se incrementa el número de patrollers reclutados con la esperanza de que algunos tengan éxito.



(a) Población de foragers. (b) Población de patrollers.

Figura 4.10: Simulación del comportamiento de las poblaciones de acuerdo a los valores de SP y UP.

Simulación 2: evaluación conjunta.

En esta segunda simulación vamos a combinar las dos poblaciones para ver cómo son capaces de organizarse la una a la otra.

- Un agente siempre tiene éxito:

$$P(\text{Éxito}|Forager) = 1$$
$$P(\text{Éxito}|Patroller) = 1$$

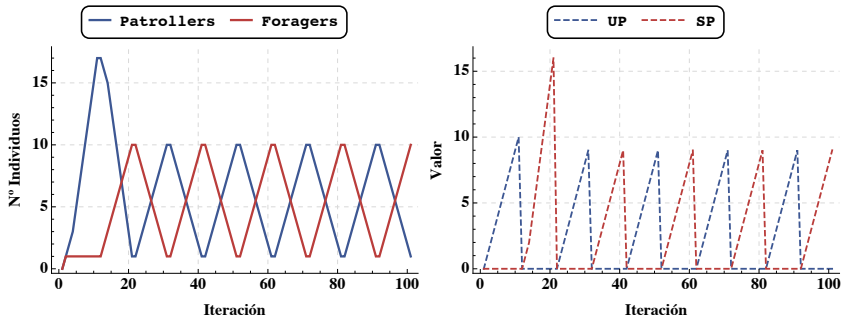
- Simulamos la señal UP:

$$UP[i+1] = UP[i] + 1$$

La señal SP viene gobernada por el número de patroller que han tenido éxito:

$$SP[i+1] = SP[i] + \text{número de patrollers que han finalizado}$$

de este modo relacionamos de manera simple la población de foragers y patrollers mediante la señal SP.



(a) Poblaciones. (b) Señales.

Figura 4.11: Simulación del comportamiento de las poblaciones combinadas. El valor de SP es función del éxito de los patrollers.

El resultado de esta simulación lo podemos ver en la figura 4.11. En este caso lo interesante es notar como aunque finalice más de un patroller por iteración, como mucho se recluta un forager. Esto impide que se genere un número excesivo de foragers. A medida que los foragers finalizan, reclutan un único patroller. Esto inhibe el reclutamiento de los patroller en función de la señal UP, lo que permite sincronizar ambas poblaciones.

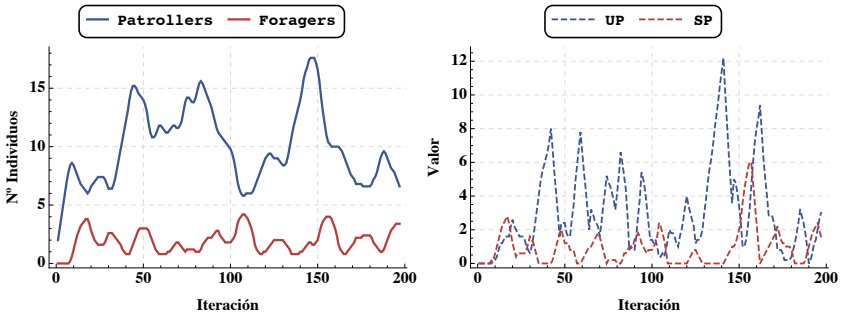
Simulación 3: evaluación realista.

En esta última simulación vamos a introducir un componente probabilístico, de tal modo que ambas señales ya no son manejadas de manera externa, sino que dependen del rendimiento de los agentes. Más concretamente:

$$P(\text{Éxito}|Forager) = 0{,}8$$
$$P(\text{Éxito}|Patroller) = 0{,}2$$

El resultado de la simulación se puede ver en la figura 4.12, en este caso al introducir probabilidades desaparecen los patrones regulares de la simulaciones previas. Como se puede ver, puntualmente aparecen picos en las poblaciones que se producen debido a situaciones donde se han acumulado una serie de fracasos en los patrollers. El sistema reacciona a esta serie incrementando el número de patrollers, lo que después se traduce en un pico en la población de foragers.

Lo más interesante de esta simulación es ver cómo las dos poblaciones se sincronizan, cuando una está en su pico más alto, la otra está en un valle. De este modo se consigue maximizar el efecto de cada población sobre la información, ya que no se generan foragers si no existe información que procesar proveniente de los patrollers, y viceversa: mientras los foragers no reduzcan la dispersión en la tabla, no interesa generar nueva información.



(a) Poblaciónes. (b) Señales.

Figura 4.12: Simulación del comportamiento de las poblaciones combinadas utilizando probabilidades.

Resumen

La inclusion de los procedimientos que se han descrito en la dinámica de población se incluyen en el prototipo diseñado en la sección anterior (3.5), para definir una versión final del algoritmo que se muestra en la figura 4.13. La implementación completa del algoritmo se puede consultar en el apéndice D.

En la tabla 4.1 se muestra una comparativa de la parametrización en función de si se incluye o no la dinámica. El algoritmo tiene tres parámetros básicos: γ_1,γ_2 y γ_3. Los dos primeros, con independencia del problema, se pueden acotar sus valores predefinidos de acuerdo con la función que cumplen (ver sección 3.5). El parámetro γ_3 es dependiente del problema, puesto que representa el número de muestras para calcular la media móvil. En mucho problemas no será necesario

```
1: procedure Bucle Principal
2:     for a ∈ Población do                                  ▷ Paso de búsqueda
3:         {L_U, P(L_U)} ← política de control
4:         acción ← selección(L_U, P(L_U))
5:         agente ← actualizar(agente)
6:     end for
7:     for a ∈ Población do                          ▷ Actualización de información
8:         if finalizado(a) == ⊤ then
9:             S ← a(S)          ▷ Obtenemos la solución a partir del agente
10:            if J(S) < μ then                                ▷ Criterio de umbral
11:                if J(S) < J_best then          ▷ Actualizamos la mejor solución
12:                    J_best ← J(S)
13:                end if
14:                μ ← actualizaMedia(S, m)
15:                λ ← (J(s)−μ)/(J_best−μ)          ▷ Calidad de la solución (Eq. 3.6).
16:                τ ← actualizaTabla(a(A_q), a(A_u), λ)
17:                exito(a)                                ▷ Dinámica de población
18:            else
19:                fracaso(a)                              ▷ Dinámica de población
20:            end if
21:        end if
22:    end for
23:    reclutamiento                                       ▷ Dinámica de población
24: end procedure
25:
```

Figura 4.13: Inclusion de la dinámica de población en el bucle principal.

usarlo ($\gamma_3 = \infty$).

Aparte de los tres parámetros básicos, existen otros dos: tamaño de la población y composición de la población, los cuales deben tener algún valor y son dependientes del problema. Además, al restringir los valores de γ_1 y γ_2, cobran mayor importancia para garantizar un balance de la búsqueda. La dinámica elimina la necesidad de ajustar estos dos parámetros, ya que son sustituidos por un esquema de auto-organización.

La auto-organización no solo permite eliminar parámetros, sino que reduce la sensibilidad al resto de parámetros. Aunque variemos el valor de los parámetros γ_1,γ_2 y γ_3, no se observara una diferencia de rendimiento en la búsqueda, ya que la población se adapta en función de la configuración paramétrica para mantener un rendimiento estable.

Por tanto, podemos decir que la dinámica de población permite eliminar parámetros dependientes del problema, a la par que reduce la sensibilidad del algoritmo[6] frente al resto de los parámetros.

[6]Un estudio de sensibilidad del algoritmo respecto de los parámetros γ_1,γ_2 y γ_3 se puede consultar en [EJGS15].

Parámetro	Valor	Prototipo	Prototipo + Dinámica
γ_1	{0,9 0,99}	✓	✓
γ_2	0.5	✓	✓
γ_3	{∞ ?}	✓	✓
Tamaño de la Pobl.	?	✓	×
Composición de la Pobl.	?	✓	×

Tabla 4.1: Comparativa de la parametrización en función de la inclusión on de la dinámica de población. Para cada parámetro se indica los valores recomendados con independencia del problema, el símbolo "?" indica que el valor debe ajustarse de acuerdo con el problema que se quiera resolver.

4.4. Análisis experimental

Una vez introducida la dinámica de población disponemos del algoritmo completo. El siguiente paso consiste en un pequeño estudio experimental para verificar empíricamente los efectos de la auto-organización.

Para realizar este estudio se va a utilizar de nuevo el problema del viajante de comercio (TSP), en las mismas condiciones que el estudio experimental anterior (sección 3.6) donde evaluamos el prototipo asíncrono. Nuevamente, las instancias que utilizaremos se toman de la librería TSPLIB'95 [Rei95], el número de ciudades viene indicado en el nombre de la instancia.

Para obtener una mejor perspectiva de los efectos de la auto-organización, se compara la versión completa, que incluye la dinámica de población, con la versión del prototipo asíncrono donde la población es fija, definida mediante parámetros: número de hormigas y porcentaje de foragers.

Análisis

Un primer experimento que hemos realizado consiste en estudiar cómo la dinámica de población afecta a la evolución de la tabla de feromona, al conjunto de soluciones generadas por el algoritmo y cómo se organiza la población. Para este estudio utilizaremos la instancia *eil*51 (51 ciudades). Para realizar la comparativa, se varía la composición de la población en el prototipo, indicando el porcentaje de foragers que componen la población de 51 hormigas.

Evolución de la entropía

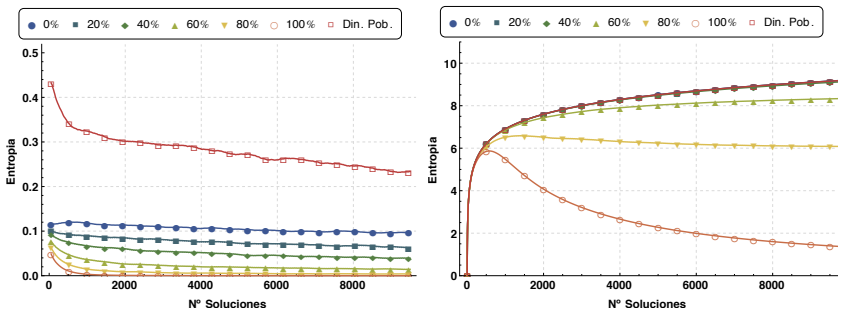
Una primera comparativa consiste en medir la evolución de la entropía, promediada utilizando 100 ejecuciones independientes. Por un lado obtenemos un promedio de la entropía media de la tabla, y el promedio de la entropía de las soluciones generadas por las hormigas.

En la figura 4.14 se puede ver el resultado de este experimento. Como se puede observar, la dinámica de población varía sustancialmente el comportamiento del algoritmo: la tabla de feromona presenta una mayor dispersión en caso de incluir la dinámica de población. Respecto a la entropía del conjunto de soluciones, si el número de patrollers es suficientemente alto, es similar se use o no la dinámica.

Esto es interesante porque indica que al incluir la dinámica, la entropía del conjunto de soluciones es alta debido al contenido de información de la tabla.

La entropía del conjunto de soluciones únicamente mide si son distintas, es decir, si hay estancamiento o no. Por tanto, si la entropía es producto de la acción de los patrollers, las diferencias entre las soluciones son producidas al azar. Por el contrario, si la entropía proviene en gran parte de la tabla de feromona, las soluciones generadas, aunque distintas, serán generadas siguiendo patrones.

Se puede esperar que las calidades de las soluciones generadas sean superiores cuando se generan utilizando la tabla, siguiendo un determinado patrón, más que puramente al azar. Es decir, el incluir la dinámica de población puede aumentar el rendimiento de la búsqueda.



(a) Entropía media de la tabla de feromona. (b) Entropía media del conjunto de soluciones.

Figura 4.14: Evolución de la tabla de feromona y el conjunto de soluciones.

Evolución de la población

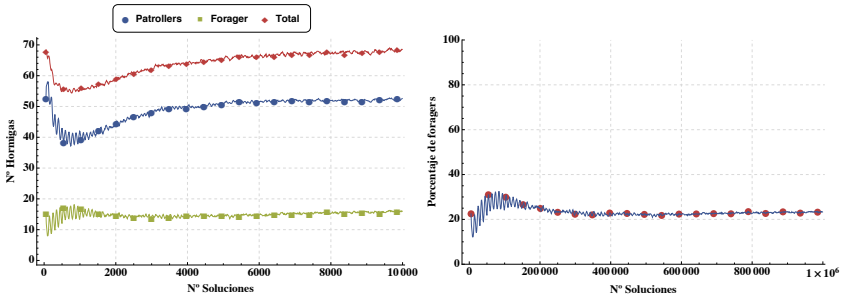
Para ver cómo funciona la dinámica de población, podemos ver cómo varía la población a lo largo de la búsqueda. Por un lado podemos medir la tendencia general, midiendo el promedio de la población en las 100 ejecuciones, y por otro lado podemos ver la evolución de la población en una ejecución aislada.

Como se puede ver en la figura 4.15, la población se organiza de manera distinta según se va avanzando en la búsqueda. La tendencia general muestra que el número de foragers se incrementa en los primeros estadios de la búsqueda para evitar una excesiva dispersión inicial, como esta configuración conduciría al estancamiento, la dinámica de población va disminuyendo el porcentaje de foragers para incrementar la exploración en situaciones más avanzadas.

La capacidad de organización de la dinámica se aprecia mejor si atendemos a una ejecución independiente. En la figura 4.16 se puede ver cómo la población se va ajustando en función del rendimiento: si el éxito de los patrollers es frecuente, se incrementa el porcentaje de foragers, aumentado su número y reduciendo el tamaño de la población. De igual modo, cuando los patrollers comienzan a fracasar, se aumenta su número para facilitar la exploración.

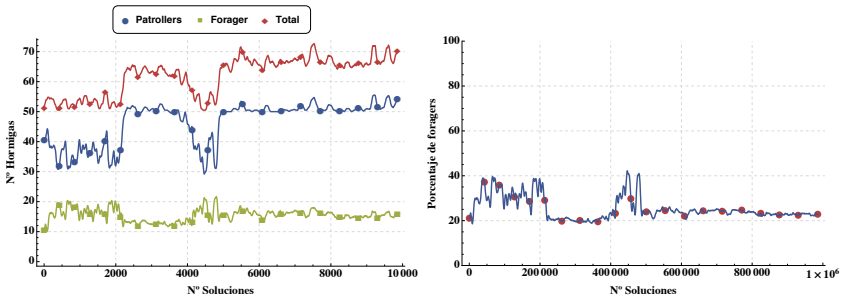
Características adaptativas

Una de las características más interesantes de la dinámica de población es justamente que se organiza en función del estado de la búsqueda, de tal modo que si variamos la representación del problema, los valores de los parámetros,



(a) Evolución de la población. (b) Evolución del porcentaje de foragers.

Figura 4.15: Auto-organización de la población a lo largo de la búsqueda, promediada en 100 ejecuciones.



(a) Evolución de la población. (b) Evolución del porcentaje de foragers.

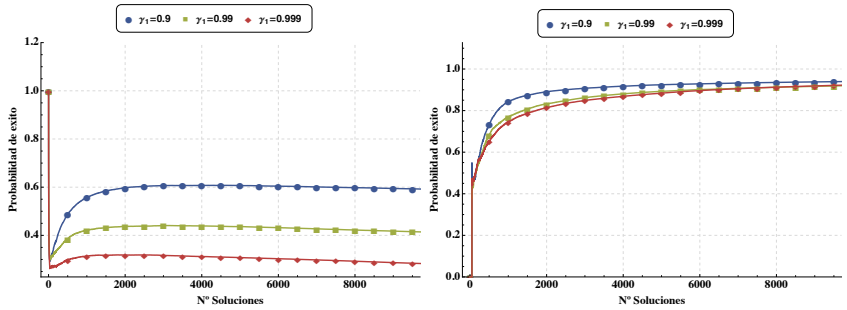
Figura 4.16: Auto-organización de la población a lo largo de la búsqueda en una ejecución independiente.

etc... la población se organizará de otra manera para obtener un buen rendimiento de acuerdo a las circunstancias de la búsqueda.

Para ilustrar esta característica podemos realizar un pequeño estudio variando el valor del parámetro $\gamma_1 \in 0{,}9, 0{,}99, 0{,}999$, cuanto mayor sea el valor de γ_1, menos probabilidades hay de que un patroller utilice la tabla de feromona para tomar una decisión.

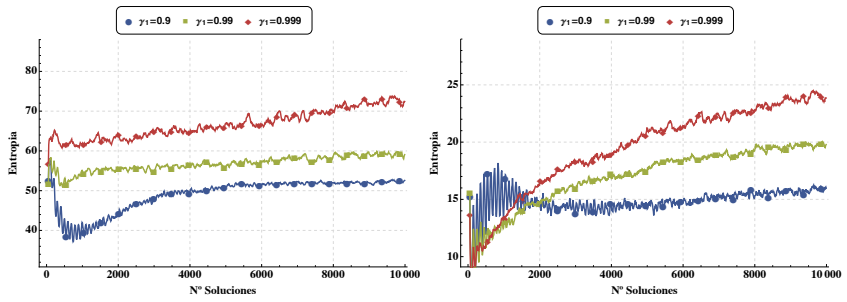
En general, al utilizar menos la tabla aumentamos la exploración del algoritmo, pero disminuimos las probabilidades de que un patroller tenga éxito. Por tanto si queremos garantizar un cierto flujo de información proveniente de los patrollers tendríamos que aumentar su número. Los foragers utilizan siempre la tabla, su probabilidad de éxito tiende asintóticamente a 1 – casi– independientemente del valor de de γ_1. En la figura 4.17 se puede ver la probabilidad de éxito tanto para patrollers como para foragers, según el valor de γ_1.

En la figura 4.18 se puede ver cómo la dinámica ajusta la población en función del valor de γ_1, disminuyendo los foragers según los patrollers tiengan menos éxito y aumentando su número cuando los patrollers tienen más éxito. En los patrollers ocurre al revés, al aumentar γ_1 incrementamos la dispersión producida por cada patroller, pero en general aciertan menos, por lo tanto es necesario aumentar su número para que la exploración global sea eficaz.



(a) Patrollers. (b) Foragers.

Figura 4.17: Evolución de la probabilidad de éxito.



(a) Patrollers. (b) Foragers.

Figura 4.18: Auto-organización de la población en función del valor de γ_1.

Evaluación

Por último, podemos medir la eficiencia real de la dinámica de población, recurriendo al error relativo cometido respecto al óptimo[7], utilizando la mismas 10 instancias que en los experimentos anteriores (sección 3.6). El error lo obtenemos calculando un promedio utilizando lotes de 100 ejecuciones.

Para comprobar que efectivamente la dinámica de población es capaz de organizar la población de manera eficiente en situaciones diferentes introducimos variaciones en forma de heurística y γ_1. De nuevo compararemos los resultados utilizando el prototipo asíncrono que utiliza tantas hormigas como ciudades tiene la instancia, variando el porcentaje de foragers desde 0 % hasta un 90 %.

- Vamos a utilizar una heurística definida como la inversa de la distancia, parametrizada según el valor de β:

$$\frac{1}{d(c_i, c_j)^\beta}$$

  donde $d(c_i, c_j)$ es la distancia entre las ciudades c_i y c_j, y $\beta \in \{0 \dots 5\}$.

- Para cada valor de β utilizamos un valor de γ_1 acorde con la heurística. Si la heurística tienen una dispersión notable, utilizamos valores de γ_1 menores,

[7] error relativo $= \frac{coste - optimo}{optimo}$

y viceversa.

β	γ_1
0	0.9
1	0.9
2	0.9
3	0.99
4	0.99
5	0.99

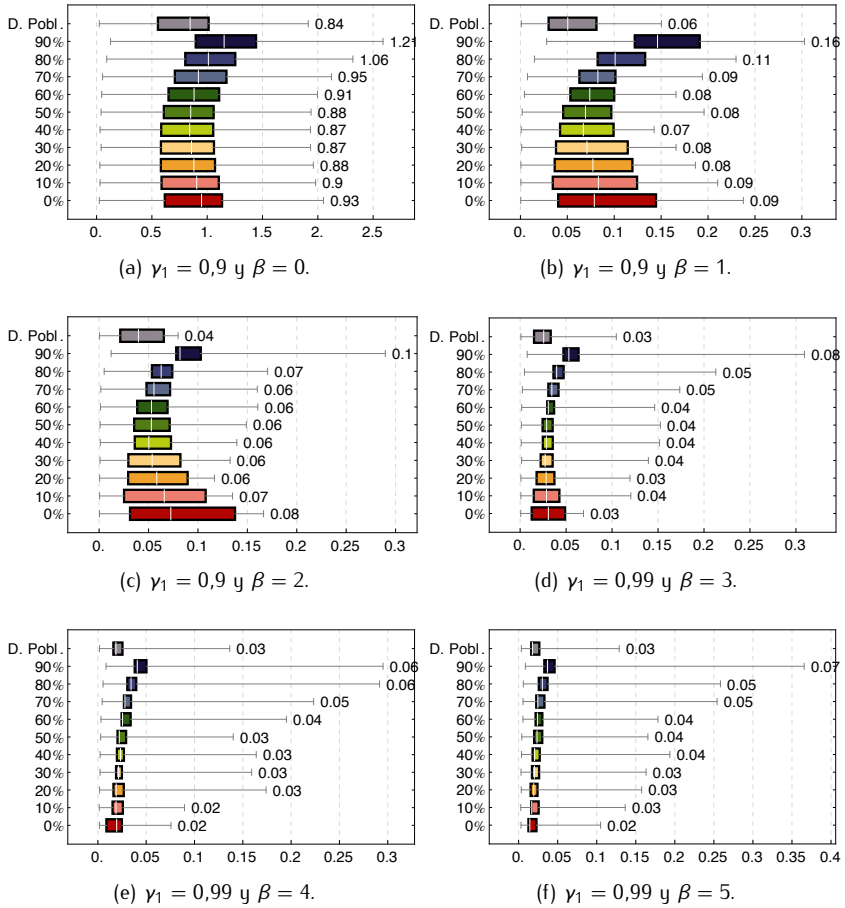
Los resultados se muestran en la figura 4.19, en la figura comparamos la diferencia de rendimiento de la dinámica con un ajuste paramétrico de la población. Como puede verse mientras la heurística no sea excesiva, el mejor rendimiento se obtiene utilizando la dinámica de población. Cuando la heurística es muy restrictiva, $\beta \gg 1$, tiene tanto peso que da prácticamente igual la población que utilicemos, ya que el rendimiento del algoritmo está muy sesgado por el uso de la heurística. Aún así, el mejor rendimiento en promedio corresponde con el uso de la dinámica de población.

La heurística del TSP es muy fiable, no es de esperar encontrar este tipo de heurísticas en otro tipo de problemas. Normalmente, las heurísticas son bastante menos fiables, lo que impide que las utilicemos de una manera tan restrictiva, donde todo el peso se acumula en unos pocos valores. Es más esperable que la heurística presente cierta dispersión, más en la linea de $\beta \in \{0 \dots 2\}$. En estos casos, la búsqueda está guiada por la heurística pero el rendimiento depende más de las características del algoritmo. En estas condiciones, podemos ver cómo la auto-organización permite un rendimiento que no se consigue mediante el uso de parámetros que mantienen una configuración fija.

4.5. Recapitulación

Inspirándonos en las dinámicas de organización de las colonias de hormigas, incluimos un procedimiento que permite a la población de agentes del algoritmo auto-organizarse. Al disponer de dos tipos de agentes podemos vincular el éxito en la búsqueda de un tipo de agente a la activación de agentes del tipo contrario. Por el contrario, el fracaso conlleva la activación de agentes del mismo tipo. De este modo se consigue reforzar un tipo de búsqueda si ésta no está teniendo éxito o la contraria si fuese necesario, manteniendo siempre una alternancia entre ambos tipos para tratar de conseguir un equilibrio.

El resultado es que el sistema es capaz de auto-organizarse sin necesidad de recurrir a parámetros para configurar la población de agentes. Como la población se adapta al estado de la búsqueda, puede comportarse de diferente manera según vaya evolucionando la búsqueda, consiguiendo una gran flexibilidad.



(a) $\gamma_1 = 0{,}9$ y $\beta = 0$.

(b) $\gamma_1 = 0{,}9$ y $\beta = 1$.

(c) $\gamma_1 = 0{,}9$ y $\beta = 2$.

(d) $\gamma_1 = 0{,}99$ y $\beta = 3$.

(e) $\gamma_1 = 0{,}99$ y $\beta = 4$.

(f) $\gamma_1 = 0{,}99$ y $\beta = 5$.

Figura 4.19: Comparativa del rendimiento relativo entre el uso de la dinámica de población y una configuración de la población mediante parámetros (porcentaje de foragers). Al final de cada diagrama de caja, se indica el valor medio del error correspondiente a cada caso.

Capítulo 5

Estudios experimentales

El algoritmo diseñado en los capítulos anteriores puede considerarse un algoritmo de hormigas. Por esta razón le hemos llamado Ant Colony Extended (ACE)[1]. Sin embargo, ACE amplía varios elementos clave de los algoritmos de hormigas:

a) Al utilizar un espacio de estados, se extiende la representación de los problemas más allá de un grafo.

b) La fuente de inspiración biológica incorpora elementos de las colonias de abejas y la organización de una colonia de hormigas. La inspiración típica de los algoritmos de hormigas se reduce a la formación de senderos de feromona.

c) Se añade una dinámica de población que permite la auto-organización de la población en función del estado de la búsqueda. De este modo el algoritmo es capaz de equilibrar la exploración de nuevas soluciones con la explotación de las ya conocidas.

En este capítulo realizaremos una serie de estudios experimentales, utilizando ACE para resolver diversos problemas. Para realizar los estudios hemos escogido tres problemas radicalmente diferentes, en vez de considerar problemas similares. De este modo se puede ver cómo los principios de auto-organización en los que esta basado el diseño del algoritmo, le permiten alcanzar un buen rendimiento en diferentes entornos de búsqueda sin necesidad de particularizar el algoritmo para cada tipo de problema, únicamente es necesario definir una representación adecuada.

Los problemas que hemos escogido son los siguientes:

i) Comparativa con algoritmos ACO clásicos utilizando el problema del TSP como banco de pruebas. El objetivo de este estudio es evaluar el rendimiento de ACE en el escenario de prueba típico de los algoritmos de hormigas.

ii) Aplicación de ACE a problemas de programación genética, comparando su rendimiento con un algoritmo genético estándar. El objetivo de este estudio es evaluar ACE en un escenario de pruebas radicalmente diferente al TSP: el desarrollo de expresiones formales.

[1] La implementación completa del algoritmo se puede consultar en el apéndice D.

Comparado con el TSP, la características más relevantes de este banco de pruebas son dos: por un lado, no existe una heurística para guiar la búsqueda; y por otro, las soluciones no tienen un tamaño prefijado, sino que éste es variable y desconocido.

iii) Aplicación de ACE a un problema real: planificación de maniobras para barcos. Este problema se diferencia de los dos anteriores fundamentalmente en que presenta una componente real. El objetivo es encontrar una serie de consignas discretas (velocidad y rumbo) que permitan al barco realizar una determinada maniobra. El calculo de la trayectoria consiste en resolver un modelo diferencial que incluye las restricciones de la dinámica del barco. Debido al coste computacional que supone la resolución numérica de un modelo diferencial, el objetivo de ACE es obtener una maniobra viable y minimizar el tiempo que requiere su realización, evaluando el modelo un número limitado de veces.

5.1. Comparativa con algoritmos ACO clásicos

Como ya se introdujo en el capítulo dos (sección 2.1), el primer algoritmo de la metodología ACO fue el AS. Dicho algoritmo fue evaluado utilizando el TSP como banco de pruebas. Aunque los resultados probaron la viabilidad de la metodología ACO, también mostraron un bajo rendimiento del algoritmo AS. Investigaciones posteriores para mejorar el rendimiento del algoritmo llevaron al desarrollo de dos algoritmos clásicos de ACO: Ant Colony System (ACS) [DG97] y Max-Min Ant System (MMAS) [SH00]. Desarrollados respectivamente por M. Dorigo y T. Stüztle.

Conviene señalar que ACS y MMAS son reconocidos como muy buenos algoritmos. De hecho, se utilizan comúnmente en el desarrollo de versiones más especializadas de algoritmos ACO para resolver otros problemas diferentes de TSP. Una revisión reciente de las modificaciones y aplicaciones de ACO se puede encontrar en [CMB12].

Aunque ACE presenta diferencias sustanciales respecto a los algoritmos que siguen la metodología ACO, es interesante realizar un estudio comparativo de ACE dentro del marco de dicha metodología. El estudio consiste en evaluar el rendimiento de ACE al resolver instancias del TSP y comparar dichos resultados con el rendimiento que ofrecen tanto ACS como MMAS.

Con el fin de facilitar la lectura, se presentará una versión simplificada del estudio comparativo llevado a cabo. Básicamente sustituiremos los detalles numéricos de los análisis estadísticos por representaciones gráficas de los mismos. El estudio completo puede consultarse en [EJGS15].

Configuración de los algoritmos

Los valores de los parámetros empleados en el TSP para ACS y MMAS son los sugeridos por la literatura [DS04]. Estos valores se muestran en la Tabla 5.1, "—" significa que el parámetro no está presente en el algoritmo. El término cl es el número de elementos en la lista de candidatos[2], L_{nn} es la longitud del

[2]Una lista de candidatos [JM97, Rei94] es una lista estática de las ciudades preferidas para ser visitadas. Para una determinada ciudad i la lista consiste en las n ciudades más cercanas.

recorrido generado utilizando la heurística del vecino más cercano, L_{mejor} es la longitud del recorrido de la mejor solución encontrada hasta el momento, y nc es el número de ciudades. En la tabla también se incluyen las ecuaciones utilizadas para establecer los límites, τ_{max} y τ_{min}, de MMAS. Por último, hay que decir que MMAS utiliza otro parámetro para regular la actualización de feromona que no está incluido en la tabla: la feromona se actualiza usando la mejor solución de la actual iteración, pero cada 25 iteraciones la feromona se actualiza utilizando la mejor solución encontrada.

Parámetro	ACS	MMAS
α	1	1
β	2	2
m	10	nc
ρ	0,1	0,02
q_0	0,9	–
ξ	0,1	–
τ_0	$1/(nc \cdot L_{nn})$	$1/(\rho \cdot L_{nn})$
cl	20	20
τ_{max}	–	$1/(\rho \cdot L_{mejor})$
τ_{min}	–	$2 \cdot \tau_{max} \cdot \frac{(1-\sqrt[nc]{0,05})}{\sqrt[nc]{0,05} \cdot (cl+1)}$

Tabla 5.1: Valores de los parámetros para ACS y MMAS.

Ambos algoritmos, ACS y MMAS fueron aplicados al TSP más con la idea de resolver el problema que para estudiar la viabilidad de la metodología utilizando el TSP como banco de pruebas. Esto implica adecuar bastante el algoritmo a las características del TSP, con lo cual el carácter de los algoritmos como metodologías de propósito general se difumina.

Para poder comparar el rendimiento de ACE con los dos algoritmos indicados es necesario adecuar la aplicación del algoritmo al problema. En primer lugar se modifica la representación del problema, en vez de definir el estado como la ciudad actual, el estado viene dado por dos ciudades: la actual y la anterior. Esto permite mantener una mayor información en la tabla de feromona acerca de posibles recorridos, ya que ampliamos el espacio de estados. Al ampliar el espacio de estados, podemos utilizar una heurística más restrictiva sin que se pierda excesiva dispersión en la búsqueda. Como heurística utilizaremos la inversa de la distancia elevada a la sexta potencia: $d(c_i, c_j)^{-6}$. Los detalles de esta representación se pueden consultar en [EJGS15].

El modificar la representación tiene como objetivo explotar de manera adecuada la heurística del vecino más cercano, la cual es clave para obtener un buen rendimiento al resolver instancias del TSP.

También se introduce como técnica auxiliar un refuerzo de la mejor solución: consiste en centrar la búsqueda en torno a la mejor solución encontrada hasta el momento. El procedimiento de refuerzo actualiza la tabla utilizando la mejor solución descubierta hasta el momento, si se acumula un cierto número de actualizaciones en la tabla utilizando soluciones cuya calidad sea inferior a 1. Igualmente los detalles se pueden consultar en [EJGS15].

Los valores de los parámetros para ACE se pueden ver en la tabla 5.2, dichos valores se obtienen de un estudio experimental que se puede consultar en [EJGS15]. Los parámetros γ_1 y γ_2 determinan el uso de la heurística por parte de

un patroller. El valor de γ_3 determina el número máximo de muestras que se utilizan para calcular la media móvil. Si se trabaja con instancias pequeñas-medianas (nº ciudades < 200) no sería necesario utilizar una media móvil. Como resolvemos instancias grandes donde el algoritmo calcula un número considerable de soluciones, es necesario utilizar la media móvil para limitar el número de muestras que utilizamos en el cálculo de la media y favorecer su convergencia. Por último, el parámetro W controla el refuerzo de la mejor solución introducido anteriormente.

Parámetro	ACE
γ_1	0,99
γ_2	0,5
γ_3	$400 \cdot nc$
W	$4 \cdot nc$

Tabla 5.2: Valores de los parámetros para ACE.

Como se puede ver comparando ambas configuraciones, existe una clara diferencia entre ACE y los dos algoritmos ACO. Al estar diseñado utilizando principios de auto-organización, ACE utiliza menos parámetros. Además, ninguno de los parámetros típicos de ACO se encuentran en ACE y viceversa.

Comparativa de rendimiento

De acuerdo con la literatura [Stu96], se sabe que AS no presenta un buen rendimiento para instancias con más de 75 ciudades. ACS y MMAS pueden resolver problemas mucho más grandes. En nuestra comparación, primero hemos resuelto instancias medianas-pequeñas de hasta 200 ciudades, cada instancia se resuelve 1000 veces. En un segundo paso, hemos tratado casos incluso más grandes, hasta 1400 ciudades, con lotes de 100 ejecuciones por instancia. En cada ejecución se permite a los algoritmos explorar como máximo $nc \cdot 10^4$ recorridos.

Una primera medida que podemos tomar es la probabilidad de encontrar el valor óptimo para cada instancia y la media del error relativo. Los resultados se muestran en las figuras 5.1, 5.2, 5.3 y 5.4. Como se puede apreciar tanto ACE como MMAS obtienen un rendimientos superior a ACS. También se puede apreciar cómo en términos generales el rendimiento de ACE es superior al de MMAS, aunque en varias instancias este último algoritmo obtiene los mejores resultados.

El rendimiento de los algoritmos lo podemos comparar de una manera mucho más precisa utilizando el test de hipótesis Wilcoxon-Mann-Whitney [GC03]. Ésta es una prueba no paramétrica de dos colas utilizada para comparar si dos poblaciones son iguales. Más específicamente, la hipótesis nula es que ambas poblaciones tienen medianas iguales. Si existe evidencia estadística de que los conjuntos de soluciones al resolver una estancia determinada de dos algoritmos no son iguales, se puede concluir que un algoritmo resuelve la instancia significativamente mejor que otro.

Los resultados de rendimiento se resumen en la figura 5.5. Esta figura muestra el ranking (cuanto más externo mejor) de cada algoritmo de acuerdo con la prueba de Wilcoxon-Mann-Whitney. Como se puede observar ACE supera a MMAS en un 72 % de los casos probados. En comparación con ACS, la mejora es del 100 %.

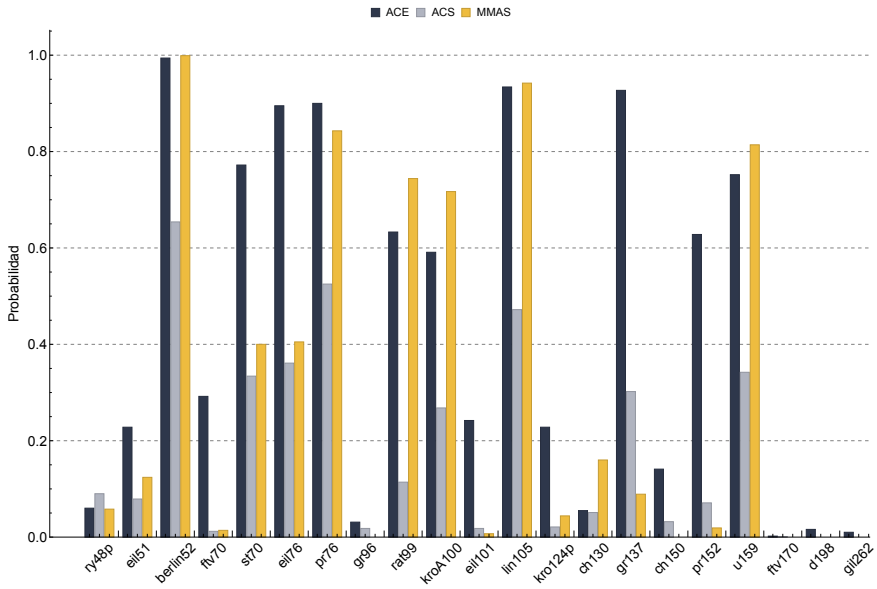


Figura 5.1: Probabilidad de encontrar la solución óptima. Para las instancias $gr431$, $d493$, $rat783$ y $fl1400$ la probabilidad es 0 para todos los algoritmos.

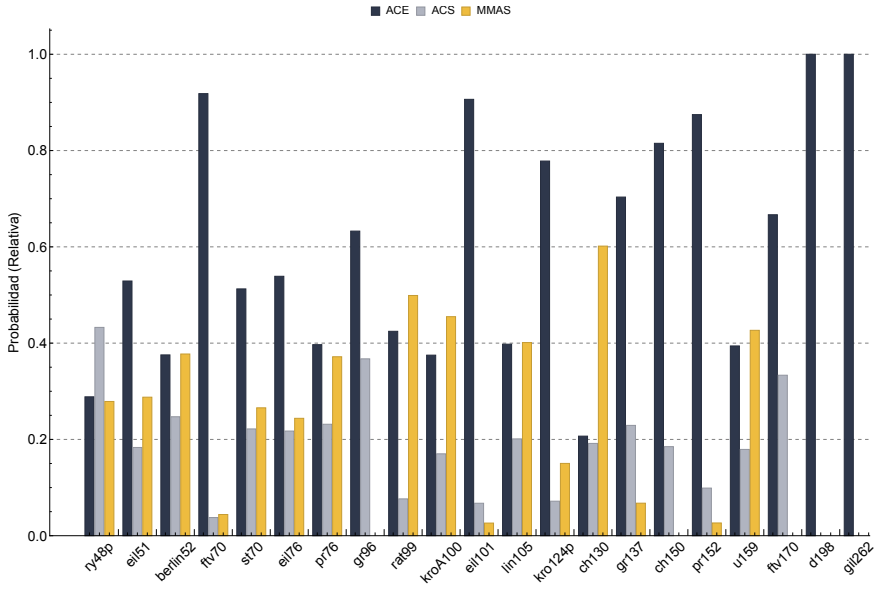


Figura 5.2: Relación entre los valores de probabilidad según la instancia: probabilidad de cada algoritmo en relación con el total (suma de probabilidades).

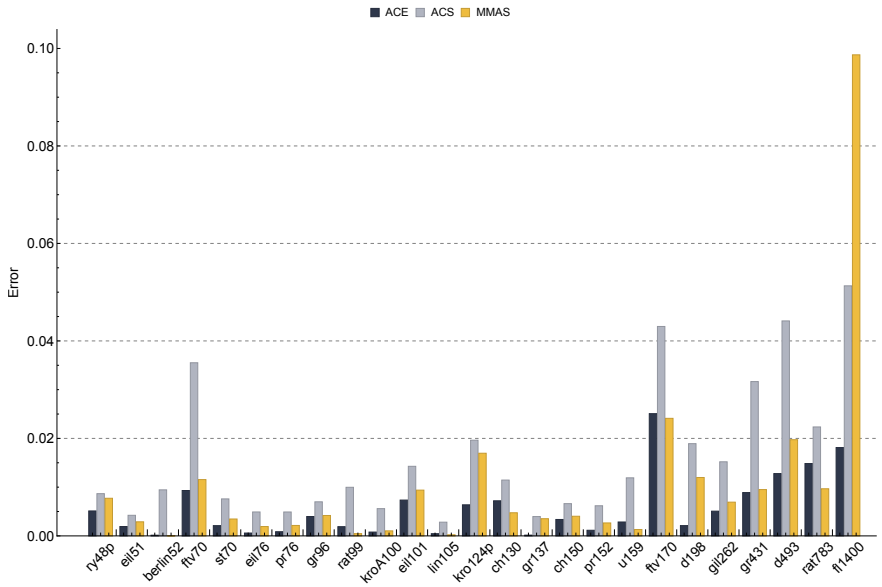


Figura 5.3: Error relativo ponderado para 1000 muestras en instancias con menos de 200 ciudades y con 100 muestras para instancias más grandes.

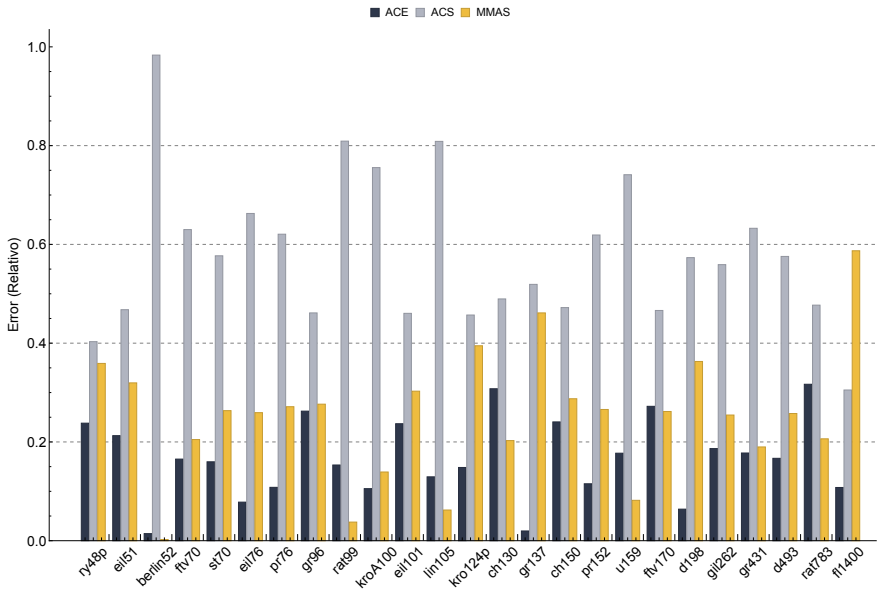


Figura 5.4: Relación entre los valores de error según la instancia: error de cada algoritmo en relación con el total (suma de errores).

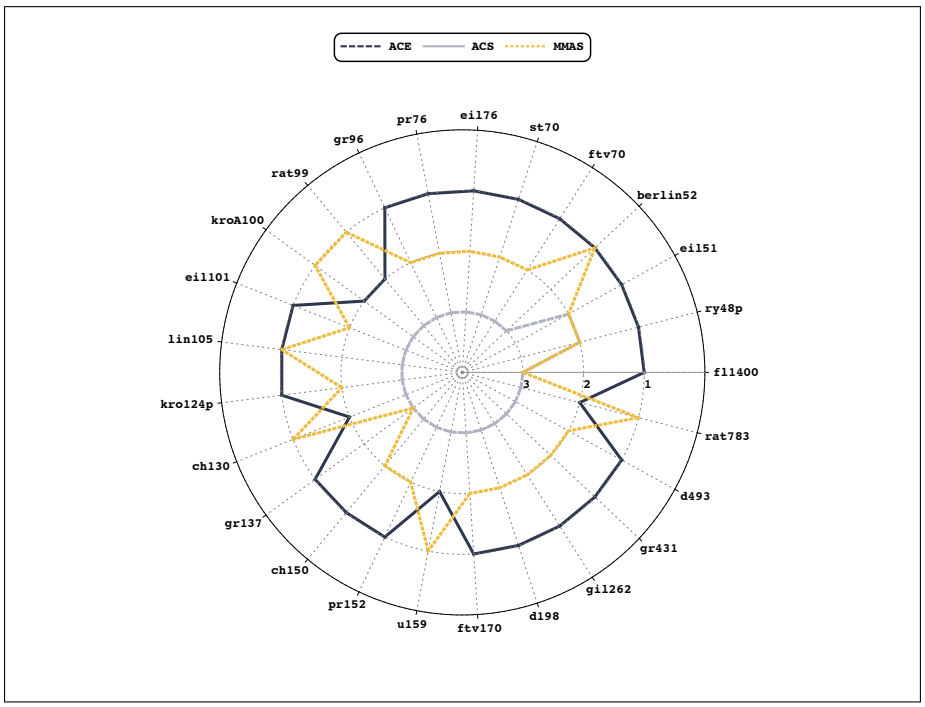


Figura 5.5: Comparativa de rendimiento de los algoritmos para cada instancia. La gráfica muestra si un algoritmo es significativamente mejor o igual que el resto de los algoritmos, de acuerdo con los resultados de la prueba de Wilcoxon-Mann-Whitney.

Esfuerzo computacional

Aparte del rendimiento que presenta cada algoritmo, también es interesante estudiar el comportamiento a lo largo de la búsqueda que ofrece cada algoritmo: si converge rápidamente hacia soluciones de calidad razonable, pero por el contrario le cuesta alcanzar soluciones próximas al óptimo, etc... Para realizar este estudio recurrimos a lo que se denomina *esfuerzo computacional* (computational effort), el cual es una medida común en problemas de programación genética [CO02, Koz92].

El esfuerzo computacional mide el número de evaluaciones de la función objetivo –el número de soluciones– que deben ser procesadas con el fin de obtener una solución de cierta calidad (δ) con una cierta probabilidad (por ejemplo, 0,99). Con esta estimación se establece cómo de eficiente es la búsqueda: cuantas menos soluciones se necesiten procesar, más eficiente será.

La comparación del esfuerzo computacional se realiza utilizando las mismas condiciones experimentales de la sección anterior. Para cada instancia, la calidad de la solución (δ) se define como una diferencia, expresada en porcentaje, respecto de la solución óptima de la instancia: 10 %, 5 %, 1 %, y 0 % (el valor óptimo). Los detalles de cómo se calcula el esfuerzo computacional pueden consultarse en el apéndice B.

Como el esfuerzo computacional es una medida que depende de la instancia, se requiere procesar más soluciones para resolver una instancia de 200 ciuda-

des que de 50, tomamos los resultados para obtener medidas relativas y poder representarlas gráficamente.

Por ejemplo, para una instancia X y un valor de δ determinado, el esfuerzo computacional de cada algoritmo es respectivamente: CE_{ACE}, CE_{ACS}, CE_{MMAS}. El esfuerzo computacional relativo de ACE (RCE_{ACE}) frente a los otros dos algoritmos lo obtenemos de la siguiente manera:

$$RCE_{ACE} = \frac{CE_{ACE}}{CE_{ACE} + CE_{ACS} + CE_{MMAS}}$$

el cálculo es idéntico para los otros algoritmos.

Los resultados del experimento se muestran en las figuras 5.6, 5.7, 5.8, y 5.9.

- Para soluciones de calidad razonable pero no excesiva, 10 % (figura 5.6) y 5 % (figura 5.7), se puede ver como ACE y ACS, por lo general, muestran una mayor eficiencia que el MMAS.

- Para obtener una solución por debajo del 1 % (figura 5.8) respecto a la solución óptima, ACS se encuentra por detrás de ACE y MMAS. El esfuerzo que requieren estos dos algoritmos es bastante similar, en algunas instancias MMAS está por debajo de ACE y viceversa.

- Cuando la solución es la óptima (figura 5.9), existen casos (las cuatro instancias con más ciudades) donde ninguno de los tres algoritmos es capaz de obtener el óptimo, pero en general, ACE es el algoritmo que requiere de menos esfuerzo para obtener una solución óptima.

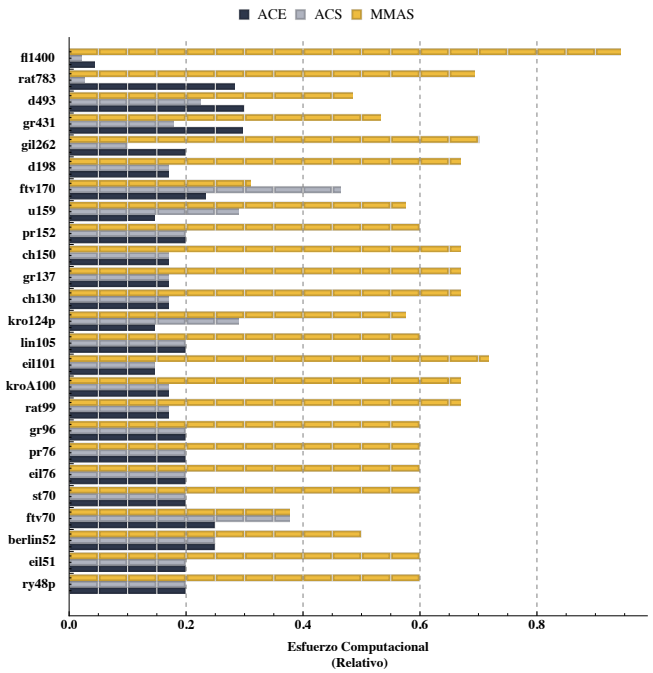


Figura 5.6: Comparativa del esfuerzo computacional para un valor de $\delta = 10\,\%$.

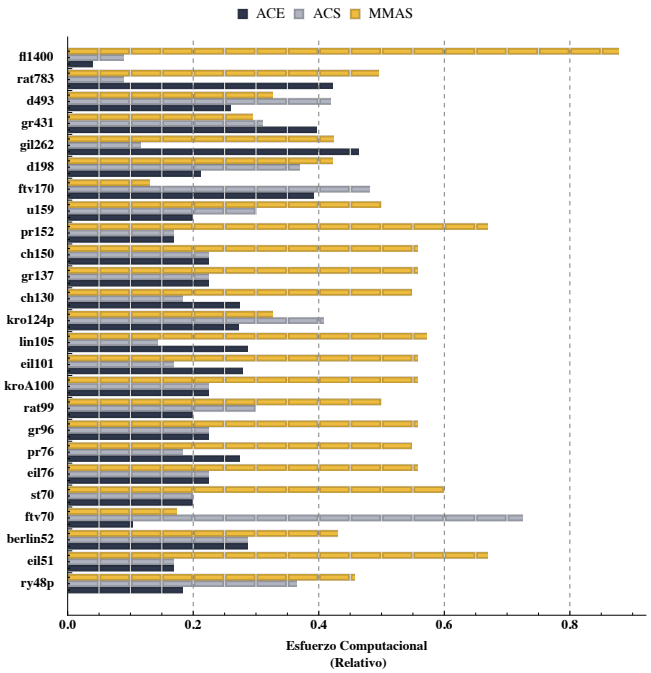


Figura 5.7: Comparativa del esfuerzo computacional para un valor de $\delta = 5\,\%$.

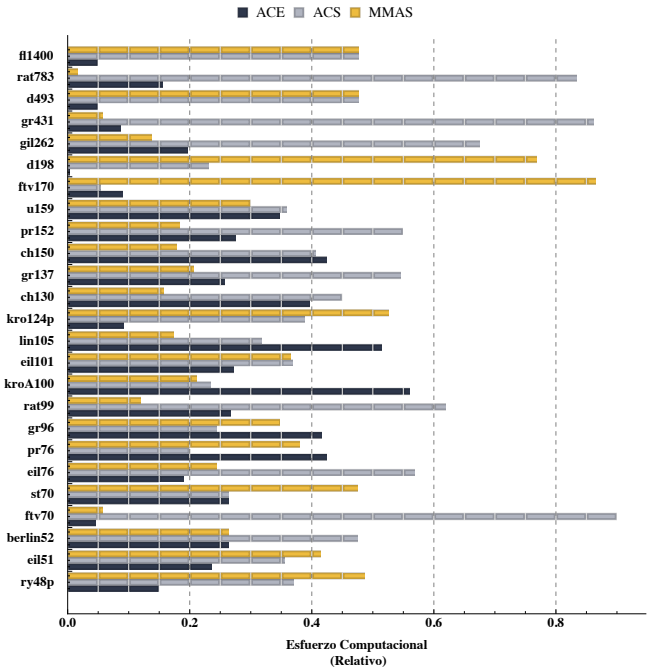


Figura 5.8: Comparativa del esfuerzo computacional para un valor de $\delta = 1\,\%$.

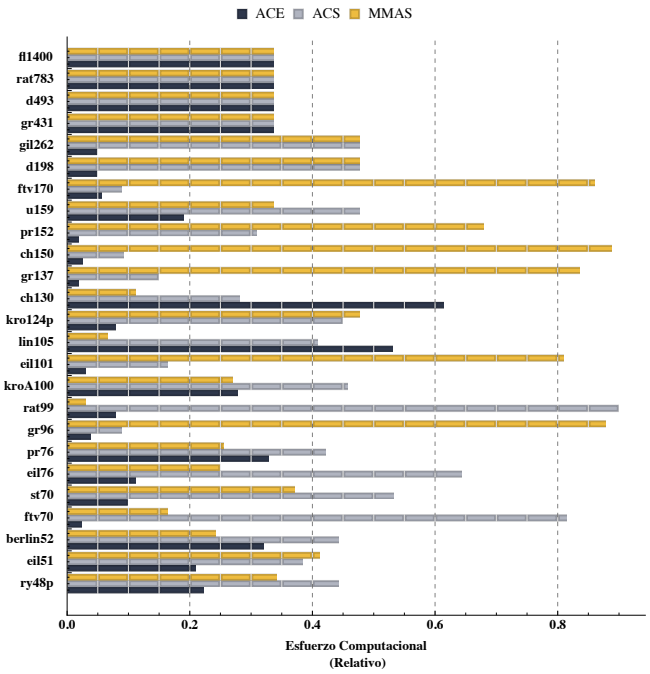


Figura 5.9: Comparativa del esfuerzo computacional para un valor de $\delta = 0\,\%$ (valor óptimo).

Es común cuando se diseña un algoritmo, si queremos obtener una solución razonable y relativamente rápido –con un número de búsquedas pequeñas– optar por un diseño que explote de manera significativa la información disponible. El peaje es que la búsqueda se estanca, el algoritmo requiere procesar una gran cantidad de información para conseguir mejorar la solución. Igualmente se puede optar por un diseño que aumente la dispersión de la búsqueda, el algoritmo no será muy rápido obteniendo soluciones razonablemente buenas, pero la calidad final mejora significativamente.

Esta es la dicotomía que aparece cuando se diseña un algoritmo. Lo más común es que el diseño final presente algún tipo de sesgo hacia uno de los dos tipos de patrones de búsqueda. Por ejemplo, ACS es un algoritmo que diversifica poco la búsqueda. MMAS es un algoritmo que la diversifica de manera significativa.

Comparativamente, ACE presenta una búsqueda mucho más equilibrada, siendo capaz de obtener soluciones de razonable calidad con un número de búsquedas reducido. A medida que se le permite procesar una mayor cantidad de información es capaz de aprovecharla de manera eficaz para mejorar la calidad de la solución. Es decir, la búsqueda no es excesivamente dispersa, pero tampoco se estanca. Esta flexibilidad se debe fundamentalmente a cómo está diseñado el algoritmo: la distribución de información entre los agentes, reforzada por el uso de la dinámica de población que permite una evolución en la organización de la población.

Comparativa temporal

Este experimento compara el tiempo consumido por cada algoritmo para resolver varias instancias. Para realizar la comparación, cada algoritmo se ejecuta 200 veces por instancia. Una sola ejecución consiste en ejecutar el algoritmo, hasta que se construye un número fijo de recorridos: $nc \cdot 10^4$, donde nc es el número de ciudades. Finalmente obtenemos el tiempo promedio en segundos necesario para resolver cada instancia.

Análogamente que al caso del esfuerzo computacional, cada instancia requiere un tiempo de cómputo diferente, normalizamos los resultados para obtener una medida relativa entre los algoritmos para cada instancia.

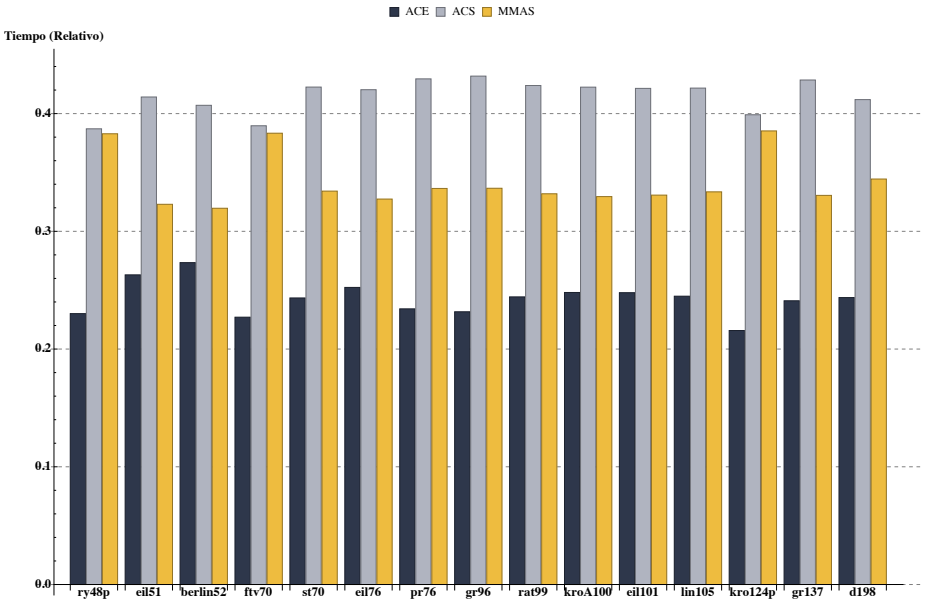


Figura 5.10: Comparativa del tiempo consumido. El valor es relativo a cada instancia: se representa un valor normalizado del tiempo que requiere cada algoritmo para resolver dicha instancia en relación con el resto de algoritmos.

La figura 5.10 muestra una comparación entre los resultados obtenidos. ACE es el que menos tiempo consume de los tres algoritmos, siendo aproximadamente 52 % más rápido que ACS, y un 33 % más rápido que MMAS. Las diferencias se deben principalmente a las diferencias en el procedimiento de construcción de soluciones y a la representación utilizada.

Como se señaló anteriormente en el texto, la tabla de feromona en ACE se inicia como una estructura vacía. Dado que el espacio de estados es desconocido, la información proporcionada por las búsquedas de las hormigas se encarga de "llenar" la tabla. Típicamente, para un cierto estado sólo hay una breve lista de operadores disponibles, haciendo el uso de esta tabla computacionalmente eficiente. En comparación, la tabla de ACO es una matriz de adyacencia de un grafo. Por lo tanto, construir una solución puede ser una operación costosa, que puede requerir el uso de técnicas auxiliares como las listas de candidatos en el caso del TSP o soluciones más refinadas como [AC07].

Concretamente para el TSP, en ACS y MMAS, las hormigas debe elegir entre una lista CL (lista de candidatos) de elementos. En ACE, cuando una hormiga utiliza la feromona elige entre 2 o 3 ciudades, las que se encuentran en ese momento en la tabla. Únicamente cuando utilizan la heurística deben escoger entre todas las ciudades de la instancia. Como la feromona se usa en el 90 % de las decisiones, en promedio, las hormigas construyen recorridos utilizando listas con menos elementos que la lista de candidatos CL.

Hibridación con operadores de búsqueda local

La hibridación con búsqueda local es una tendencia en la aplicación de los algoritmos ACO [Stu97, DS04] y en el uso de meta-heurísticas en general [HS04].

De manera genérica los podemos definir como operadores de búsqueda exhaustiva o ciega que permiten la exploración de un conjunto de soluciones según un determinado patrón. Es decir, a partir de una solución se genera un conjunto de soluciones, realizando pequeñas modificaciones sucesivas a la solución original.

Un operador de búsqueda local combinado con un algoritmo de orden superior como una meta-heurística aumenta el volumen de información procesada por la meta-heurística, lo que obviamente conlleva un incremento en el rendimiento del algoritmo. En parte, el éxito de la búsqueda local está ligado al aumento de la potencia de cómputo de los ordenadores, lo cual ha permitido el uso de técnicas de búsquedas pseudo-exhaustivas que con una potencia de cómputo limitada serían inviables.

Es común presentar los operadores de búsqueda local como una generalización de los algoritmos ACO. Hay que advertir que ACO se define como un procedimiento general, mientras que el operador de búsqueda local puede no serlo. Por ejemplo, los operadores locales del TSP *2-opt*, *3-opt*, *Lin–Kernighan* [JM97] son métodos heurísticos ad hoc para resolver el TSP.

Para los experimentos de hibridación con búsqueda local utilizaremos el operador $3-opt$ [JM97]. Con el fin de realizar la comparación, se seleccionaron varios casos de instancias grandes del TSP. Los detalles del estudio se pueden encontrar en [EJGS15].

Si queremos combinar adecuadamente los algoritmos con el operador de búsqueda local, es necesario modificar el MMAS de tal modo que la actualización de la feromona sea siempre con la mejor solución descubierta hasta el momento. Igualmente, ACE también debe ser modificado. En la misma línea, la modificación consiste en permitir la actualización de la feromona únicamente para aquellas hormigas con una calidad de solución igual a la mejor solución descubierta hasta el momento. Como se puede ver, para ambos algoritmos la modificación consiste en reducir la dispersión de la búsqueda, el operador de búsqueda local ya dispersa suficiente, con lo cual si no se modifican los algoritmos, el rendimiento general decae ya que la dispersión global es excesiva.

Los resultados se muestran en las figuras 5.11 y 5.12, donde se compara el rendimiento de los algoritmos de acuerdo con los resultados de la prueba de Wilcoxon-Mann-Whitney. Cuando el número de soluciones exploradas permitido es bajo ($100 \cdot nc$), ACE es mejor que MMAS en todos los casos probados, y mejor que ACS en 6 de los 10 instancias. Cuando aumentamos el número de soluciones construidas, MMAS es capaz de lograr el mismo rendimiento que ACE en casi todos los casos, mejorando los resultados de ACS.

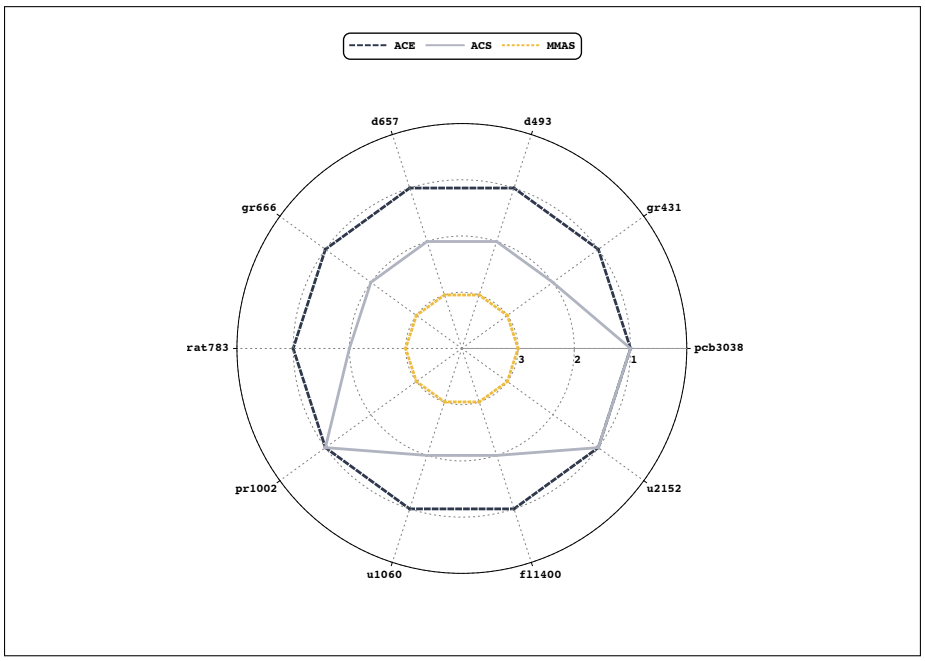


Figura 5.11: Comparativa de la hibridación con búsqueda local, permitiendo construir $100 \cdot nc$ soluciones.

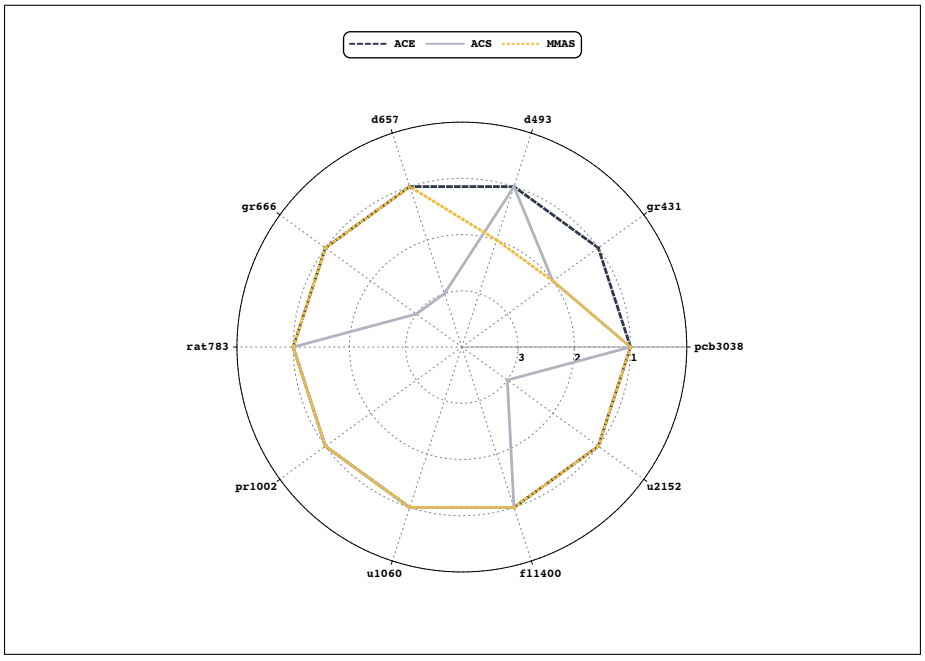


Figura 5.12: Comparativa de la hibridación con búsqueda local, permitiendo construir $5000 \cdot nc$ soluciones.

Aunque hay diferencias pequeñas en función de la instancia, los algoritmos tienen un rendimiento similar. Esto se debe a que las capacidades de búsqueda de cada algoritmo se ven eclipsados por el procedimiento de búsqueda local que domina la búsqueda. De hecho para el TSP, los operadores locales son tan efectivos que no necesitan combinarse con ninguna meta-heurística, por si solos resuelven el problema. Solamente hay que aplicarlos dentro de un esquema iterativo sencillo.

5.2. Programación genética

La programación genética (GP) [LP02, Koz92] fue el término acuñado por Koza para denominar al campo que consiste en el desarrollo de expresiones formales mediante el uso de algoritmos genéticos. Aunque teóricamente se puede considerar como una aplicación de un algoritmo genético, dicho algoritmo está tan particularizado para este tipo de problemas que se considera como un campo de estudio en sí mismo.

El objetivo de los problemas de programación genética es obtener una expresión de acuerdo con los elementos de una gramática formal G. En general, una gramática puede ser especificada por un conjunto de términos functores G_F, término cuya aridad (número de argumentos) es mayor que cero, y un conjunto de símbolos terminales G_T, cuya aridad es cero. Las expresiones se representan como árboles ordenados con una raíz única[3], compuestos por una serie de nodos y hojas. Los nodos representan términos funcionales de la gramática y las hojas los símbolos terminales.

Por ejemplo, supongamos que disponemos de una gramática con los operadores lógicas and y or, como son operadores binarios (aridad dos) utilizamos dos símbolos terminales que representen sus argumentos: $D0$ y $D1$. Podríamos especificar[4] la gramática de la siguiente manera:

$$G_F = AND|OR$$
$$G_T = D1|D0$$

Utilizando esta gramática podemos desarrollar infinitas expresiones lógicas, como la que se muestra en la figura 5.13: $AND(OR(D0, D1), D0)$. Como se puede intuir las expresiones se forman uniendo nodos funcionales entre sí. Las hojas se "rellenan" con símbolos terminales.

Obviamente dada una gramática, el objetivo es obtener una expresión de un cierto tipo. Es decir, que satisfaga algún predicado. En programación genética es común encontrar el uso del término predicado como el requisito a satisfacer para dar por buena una expresión como solución del problema. Para lo cual se utiliza una función objetivo J que permite evaluar una expresión y determinar cual es el error cometido frente al predicado que se desea satisfacer.

Por ejemplo, si estamos buscando una expresión lógica es normal recurrir a la tabla de verdad: para cada valor de las variables se especifica el valor de salida

[3] En aras de la legibilidad, un árbol ordenado de raíz única se denominará simplemente árbol.

[4] Siendo estrictos la gramática no está bien especificada como gramática formal (Véase por ejemplo [BN99]), pero para este contexto consideraremos una gramática especificada correctamente simplemente mediante la enumeración del conjunto de símbolos funcionales y terminales.

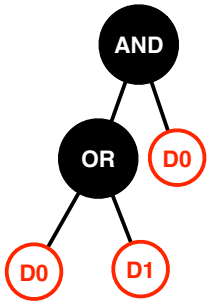


Figura 5.13: Ejemplo sencillo de una expresión lógica compuesta por símbolos *and* y *or*.

de la expresión. Por ejemplo la tabla de verdad de la expresión XOR (O exclusiva) es la siguiente:

A	B	Z
$\bot$	$\bot$	$\bot$
$\bot$	$\top$	$\top$
$\top$	$\bot$	$\top$
$\top$	$\top$	$\bot$

Si quisiésemos evaluar la expresión de la figura 5.13 para ver si se ajusta a la tabla de verdad de la expresión XOR, simplemente tendríamos que ir dando valores la los términos $D0$ y $D1$ y evaluar la expresión:

$D0$	$D1$	Z
$\bot$	$\bot$	$\bot$
$\bot$	$\top$	$\bot$
$\top$	$\bot$	$\top$
$\top$	$\top$	$\top$

Como se puede ver únicamente coinciden 2 combinaciones posibles de las 4 que definen la tabla de la función XOR, por tanto el error cometido es del 50 % (2/4).

Al utilizar una función objetivo para medir el error de una expresión, es posible desarrollar expresiones de manera automática utilizando un algoritmo de búsqueda meta-heurístico. El algoritmo explora posibilidades, posibles expresiones. Utilizando la minimización del error como guía, se obtiene un conocimiento acerca de qué estructuras son más idóneas para satisfacer el predicado.

Este tipo de problemas constituyen un marco de estudio muy atractivo ya que las gramáticas formales son, en cierta manera, la base de la computación. Mediante una gramática formal podemos desarrollar expresiones lógicas, expresiones matemáticas e incluso programas de ordenador si la gramática provista representa un lenguaje de programación. Como se puede intuir existen dos dificultades fundamentales:

a) Dada una gramática el conjunto posible de expresiones que podemos generar es infinito, luego para predicados que requieran expresiones complejas o gramáticas grandes una búsqueda directa no es posible si se desea obtener resultados aplicables. Es necesario aproximar el problema de otra manera.

b) La expresión generada mediante un algoritmo puede ser "extraña" desde la perspectiva humana. Nosotros tendemos a generar expresiones simplificadas y legibles, un algoritmo no tiene tal capacidad, simplemente va combinando fragmentos de expresiones hasta que satisface el predicado, por tanto la expresión resultante suele ser bastante caótica.

Aunque en este contexto únicamente nos vamos a plantear ejemplo sencillos, la programación genética forma parte de un área interesante conocida como diseño evolutivo. En muchos casos los diseños originales están realizados por humanos y responden a un cierto conocimiento "intuitivo". Algunos diseños ni se plantean porque "no tienen sentido". En cambio, una técnica evolutiva los prueba todos, los que para nosotros tienen sentido y los que no. Si bien es cierto que la intuición humana es bastante fiable, existen ejemplos donde el diseño automático mediante técnicas evolutivas ha conseguido obtener resultados que un humano no se hubiera planteado por considerarlos absurdos [LHL05].

En este segundo estudio experimental nos proponemos aplicar ACE a diversos problemas tipo de los originales utilizados por Koza. El objetivo de este estudio tiene un doble propósito, primero evaluar el algoritmo fuera de los problemas tipo de ACO, y ver cómo la forma de buscar de ACE puede ayudar a obtener expresiones más simples de manera automática.

En la literatura del ACO apenas existen unos pocos trabajos preliminares para resolver problemas de programación genética. Uno de los primeros trabajos publicados es la aproximación del *AntTag* [AHM02], el estudio se centra bastante en cómo aplicar la metodología al problema, pero carece de un estudio experimental adecuado para obtener alguna conclusión. Otro trabajo que merece la pena destacar es el llamado "Enhanced Generalized Ant Programming" [SAW08] que se basa en un trabajo previo [KS02]. Este segundo trabajo es más completo que el anterior, pero aún así el análisis experimental tampoco es concluyente, ya que utiliza versiones simplificadas de los problemas originales de Koza. A día de hoy, no existe un estudio experimentalmente riguroso que utilice ACO para resolver problemas de programación genética como para considerar una posible comparación con ACE.

Representación del problema

El uso de un algoritmo genético implica que se dispone inicialmente de una colección de expresiones previamente generadas, las cuales se combinan o se modifican para formar expresiones complejas. En ACE no se dispone de soluciones o expresiones completas, sino que cada hormiga debe ser capaz de construir una solución. Por tanto un primer paso para obtener una representación adecuada del problema consiste en definir cómo se construye una expresión en forma de árbol.

Como la búsqueda de una hormiga puede detenerse en cualquier momento, el resultado de realizar un paso de búsqueda debe ser siempre una expresión que pueda ser evaluada. Esto se conoce como una representación robusta [Koz92]. Para construir un árbol de manera robusta podemos realizar el siguiente proceso:

1. Generar un árbol simple compuesto por un único nodo funcional y tantos terminales como la aridad del functor utilizado. Este tipo de árboles los llamaremos árboles atómicos para denominar aquellos árboles que no pueden ser descompuestos en otros más simples[5].

[5] No estamos considerando como árboles aquellos compuestos por un único nodo.

2. Un nuevo árbol se genera a partir del árbol anterior sustituyendo un nodo terminal por un árbol atómico.

3. Repetimos el paso 2 tantas veces como sea necesario para construir un árbol complejo.

En la figura 5.14 se ilustra un ejemplo sencillo de este procedimiento de construcción utilizando *árboles atómicos*. Como se puede ver el árbol va aumentado de complejidad a medida que vamos realizando acciones: sustituir un nodo terminal por un árbol atómico. Siempre, en todos los pasos intermedios, se generan árboles que pueden ser evaluados. Una vez que hemos definido como se construye un árbol, hay que definir una representación en un espacio de estados adecuada para ACE.

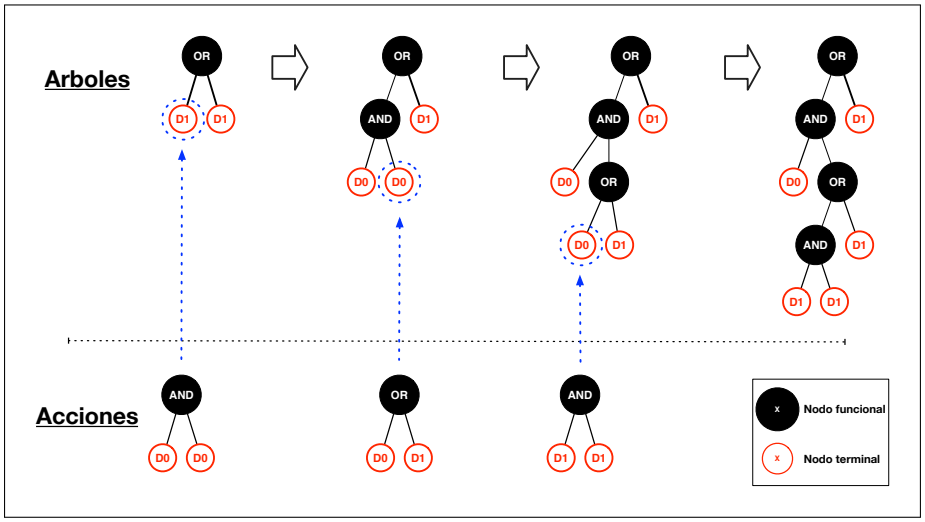


Figura 5.14: Ejemplo de construcción de un árbol complejo utilizando árboles atómicos (árboles completos que únicamente contienen un nodo funcional).

Representación adecuada

Con representación adecuada nos referimos a una que permita utilizar el conocimiento obtenido en la construcción de un árbol para construir otros posteriormente, es decir *permitir el aprendizaje*.

En nuestro caso el aprendizaje se basa en asociaciones estado-acción. Esta forma de aprendizaje plantea un problema cuando *el estado es tan específico que dificulta reutilizar el conocimiento*. Es decir, para poder repetir una acción aprendida previamente, el estado actual debe coincidir *exactamente* con un estado previo. Si los estados son muy específicos, contienen mucha información, este encaje es muy difícil que se de y el aprendizaje se puede volver inviable.

Este problema se puede solventar si se utilizan patrones locales referidos a una situación global para definir los estados. El encaje de un patrón local es relativamente sencillo, por lo que se puede repetir la acción asociada a él en multitud de situaciones. De este modo se facilita la reutilización del conocimiento adquirido, mejorando la viabilidad del aprendizaje.

En el apéndice C se puede consultar un ejemplo detallado acerca de esta problemática y el uso de patrones locales como forma de solución.

En el caso de los árboles de expresión, para evitar este problema definimos los estados como sub-árboles relativos al árbol de expresión que estamos construyendo. Como un árbol puede contener multitud de sub-árboles, definimos un tipo de acción que nos permite cambiar de estado escogiendo otro sub-árbol distinto al actual, pero sin modificar el árbol de expresión que estamos construyendo.

Una vez introducido el tipo de representación, podemos detallar el procedimiento de construcción de expresiones.

Construcción de expresiones

El proceso se inicializa a partir de un árbol atómico dado. Este árbol atómico pasa a ser el árbol de expresión que estamos generando.

- Definimos los estados como patrones simples a partir del árbol de expresión que estamos generado. La forma más sencilla es utilizar sub-árboles con una profundidad máxima L. De este modo si L es suficientemente pequeña los sub-árboles son sencillos y los árboles complejos pueden compartir multitud de sub-árboles.

- Las acciones consisten en seleccionar un nodo del sub-árbol distinto de la raíz y un árbol atómico.

- La transición entre dos estados depende del nodo seleccionado en la acción:

  a) Si el nodo es funcional, se pasa a un nuevo estado utilizando el sub-árbol que tiene como raíz el nodo seleccionado. El árbol atómico no se utiliza y se descarta.

  b) Si el nodo es terminal, se sustituye por el árbol atómico escogido. Como el árbol de expresión ha cambiado, el estado siguiente se construye como el sub-árbol definido a partir de la raíz del árbol de expresión.

Como se puede ver el proceso es muy simple vamos moviéndonos por el árbol desde la raíz hacia abajo a partir de sub-árboles con una profundidad restringida. Cuando realizamos una sustitución de un terminal por un árbol atómico, volvemos a la raíz para comenzar de nuevo.

El proceso de búsqueda se ilustra en la figura 5.15. Los estados son árboles de profundidad 2 como máximo, pero pueden ser más pequeños. Como se puede apreciar partimos de un árbol atómico que sirve de inicialización para la búsqueda.

i) El primer paso consiste en seleccionar del estado actual un nodo "distinto de la raíz". Como solo hay nodos terminales, se inserta un nuevo árbol atómico donde exista un terminal. Hemos realizado una sustitución, por tanto el estado actual viene dado por el nodo raíz del árbol de expresión.

ii) En el siguiente paso volvemos a elegir un nodo del estado actual. En este caso se trata de un nodo funcional, por tanto actualizamos el estado seleccionando como raíz del sub-árbol el nodo seleccionado. El árbol atómico se descarta puesto que no existe sustitución.

iii) Nuevamente nos encontramos en un estados cuyos únicos nodos son terminales, por tanto realizamos una sustitución del nodo seleccionado. El nuevo estado actual está determinado por la raíz del árbol de expresión.

Para cada estado, la acción puede ser escogida al azar o bien estar contenida en la tabla de feromona. Cada estado –cada sub-árbol– se asocia con la acción realizada: el nodo seleccionado y el árbol atómico utilizado. De este modo podemos repetir la misma acción cuando nos encontramos con el mismo sub-árbol.

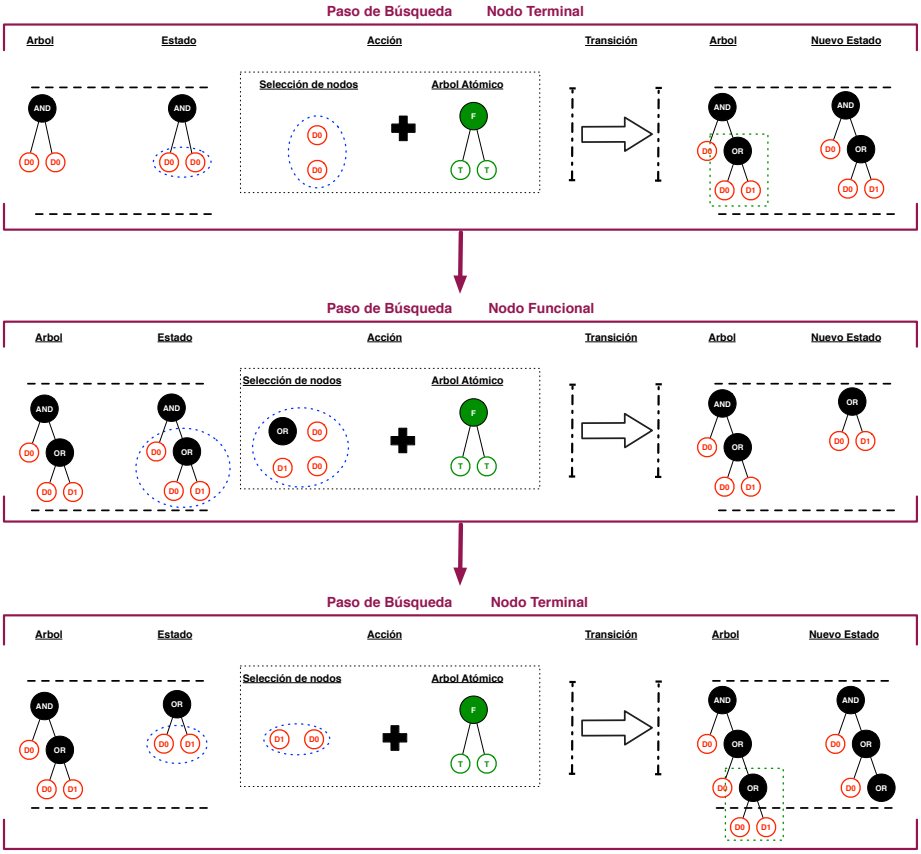


Figura 5.15: Ejemplo sencillo de un proceso de búsqueda para la construcción de un árbol de expresión.

Esta representación permite almacenar conocimiento acerca de patrones sencillos de construcción que pueden usarse en la generación de diferentes árboles de expresión. Así permitimos al algoritmo que aprenda cómo construir las expresiones más adecuadas de acuerdo con la gramática y el predicado pedido.

Aparte del grueso de la representación tal y como se ha descrito, existen otros detalles que merece la pena considerar con más detenimiento.

Tabla de feromona

En la tabla almacenamos como estado sub-árboles de profundidad L. A cada estado se le asocia una lista de posibles acciones cada una con un peso para

determinar su elección.

Cada acción consiste en un nodo del sub-árbol (del estado) que determina el nodo que se debe seleccionar. Si dicho nodo es terminal, la acción incluye también el árbol atómico para realizar la sustitución. Si es funcional, no se incluye árbol atómico puesto que no se utiliza.

El esquema de tabla se ilustra en la figura 5.16.

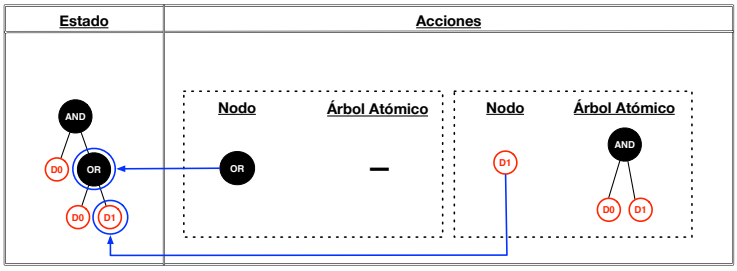


Figura 5.16: Esquema del contenido de la tabla de feromona.

Estado inicial

Para inicializar la construcción es necesario un primer árbol atómico. Definimos un estado inicial (q_0) que representa el estado previo a que se seleccione ese primer árbol atómico. De este modo, este primer árbol se puede escoger al azar utilizando la heurística, pero también se puede seleccionar utilizando la feromona: accediendo al contenido del estado q_0.

Criterio de parada

Para establecer el criterio de parada en la construcción de un árbol recurrimos a un símbolo funcional especial: END que tiene aridad 0. Como tiene aridad 0 nunca va a formar parte de una posible gramática.

Cuando se selecciona una acción cuyo árbol atómico está formado por un único nodo funcional END, es la señal que utiliza una hormiga para dar por finalizada la construcción del árbol (el proceso de búsqueda) y pasar a evaluar la expresión generada. Por tanto, si el problema viene especificado por un conjunto de símbolos funcionales G_F, ACE trabaja con un conjunto ampliado $G'_F : G_F \cup \{END\}$

Al definir la parada como una acción (el símbolo END debe formar parte de un árbol atómico seleccionado en un acción), el propio algoritmo aprende cúando debe detener la construcción de un árbol. Recordemos que en la tabla de feromona almacenamos asociaciones-estado acción, por tanto la acción de parada se almacena como una acción más vinculada a un estado. De este modo, en sucesivas construcciones se puede saber cuándo detener la generación del árbol.

Elección heurística

Como en este caso parte de las acciones son elementos de una colección finita de árboles atómicos, podemos evaluarlos previamente a iniciar la búsqueda y de este modo obtener un distribución de qué árboles atómicos conviene utilizar más a menudo.

Más formalmente, para elegir al azar una acción se deben tomar dos decisiones:

Selección de nodo La selección de un nodo v se realiza de acuerdo a la siguientes ecuaciones:

$$P_\eta(v|q) = \frac{1}{|L_q|} \tag{5.1}$$

El término $P_\eta(v|q)$ es la probabilidad de elegir el nodo v en el estado q, L_q es el conjunto de nodos contenidos en el sub-árbol que representa el estado q excluyendo el nodo raíz, $|L_q|$ es el número de elementos de dicho conjunto.

Como se puede ver es simplemente una elección al azar sobre el conjunto de nodos diferentes a la raíz que componen un sub-árbol.

Selección de un árbol atómico La selección de un árbol atómico e_1 se realiza de acuerdo con la siguientes ecuaciones:

$$P_\eta(e_1|q) = \frac{\eta(e_1, q)}{\sum_{e'_1 \in E_1} \eta(e'_1, q)} \tag{5.2}$$

donde

$$\eta(e_1, q) = \frac{1}{J(e_1)} \tag{5.3}$$

El término E_1 es el conjunto de todos los posibles árboles atómicos que se pueden construir, $|E_1|$ es la cardinalidad de dicho conjunto. El término $J(e_1)$ es el error asociado al árbol atómico e_1.

En este caso cada árbol atómico tiene asociada una probabilidad de ser escogido en función del error cometido al utilizarlo de manera aislada como solución al problema.

La única salvedad es el símbolo END, al tratarse de un símbolo auxiliar no puede ser evaluado. Para solucionarlo se le asocia una probabilidad *a priori*, P_{END}, la cual se utiliza para determinar cuando una hormiga que realiza una búsqueda heurística, va a generar un árbol atómico de parada o bien un árbol atómico con el resto de símbolos de la gramática. Este procedimiento se describe en pseudo-código en la figura 5.17.

```
1:  procedure HEURÍSTICA(ant)
2:      R ← random(0, 1)
3:      if R > P_END then
4:          acción = [∅, END]
5:      else
6:          nodo ← seleccionNodo()
7:          atómico ← seleccionAtomico()
8:          acción = [nodo, atómico]
9:      end if
10:     ants ← actualiza(acción)
11: end procedure
```

Figura 5.17: Descripción en pseudo-código del procedimiento del uso del símbolo END para detener una búsqueda cuando se utiliza la heurística.

Experimentos

Vamos a realizar dos bloques de experimentos. Un primer bloque consiste en un conjunto de problemas tipo definidos por Koza [Koz92], en los cuales la función de evaluación está definida sobre los números enteros, es discreta. Para este bloque utilizaremos una librería de solución de problemas de programación genética (ECJ) [LPB+06] para resolver los problemas con un algoritmo genético básico y obtener así una comparativa.

El segundo bloque consiste en utilizar una gramática matemática para realizar regresiones de funciones. Se utiliza una función de evaluación que mide un error real y es por tanto contínua. En este segundo bloque utilizaremos resultados de la literatura para realizar la comparativa.

Problemas discretos

Los estudios experimentales consisten en medir el esfuerzo computacional que requiere cada algoritmo para resolver el problema, es decir, el número de expresiones necesarias para garantizar con un 99 % que se satisface el predicado de cada problema.

Como ya se introdujo en los experimentos del TSP, el esfuerzo computacional [Koz92] es un medida típica de programación genética que precisa estimar la probabilidad de encontrar el óptimo según el número de expresiones evaluadas. Esta estimación se hace experimentalmente y no es fiable si el número de muestras es reducido. Por lo tanto, para estimar el esfuerzo computacional ejecutamos cada algoritmo 1000 veces permitiéndole evaluar hasta 1×10^6 expresiones. Si el predicado se satisface antes de haber consumido el número máximo de evaluaciones permitida, la búsqueda se detiene.

Una segunda medida que vamos a tomar es el número de nodos que componen aquellas expresiones que satisfacen el predicado. Tomamos este segundo dato porque en programación genética existe un problema que se conoce como "Bloat" [Pol03, LP06] que hace referencia al tamaño de la expresión generada. Al igual que en la naturaleza el código genético está lleno de material inservible, fragmentos que se han ido acumulando a lo largo de la evolución, en programación genética ocurre algo parecido, se acumulan fragmentos de expresiones sin ninguna funcionalidad, términos que se cancelan entre sí, etc... Al final aunque el algoritmo dé con la solución, la calidad y la legibilidad de la expresión propuesta puede ser bastante pobre. Una forma sencilla de estimar la calidad de una expresión es contabilizar su número de nodos.

El Algoritmo Genético (GA) lo parametrizamos con los valores por defecto que vienen indicados en [Koz92], únicamente incluimos los valores del tamaño de la población y el número de generaciones, que se han aumentado para permitir al algoritmo evaluar 1×10^6 expresiones antes de finalizar la ejecución. En el caso de ACE los valores de los parámetros se detallan en la tabla 5.3.

Artificial Ant Trail

Este problema consiste en encontrar una expresión que gobierne un hipotético robot. El objetivo es que el robot encuentre diferentes elementos (piezas de comida) distribuidos en un mallado. El robot puede orientarse en cuatro direcciones: norte,

Parámetro	ACE
γ_1	0,9
γ_2	0,5
γ_3	∞
L	2
P_{END}	0,95

Parámetro	GA
Población M	1000
Generaciones G	1000

Tabla 5.3: Valores de los parámetros para el algoritmo genético y ACE.

sur, este, y oeste. Únicamente puede avanzar (una casilla) en la dirección hacia donde está orientado.

Existen dos escenarios posibles: "Santa Fe trail" y "Los Altos trail", con 89 y 157 piezas respectivamente. Obviamente el segundo es más grande y representa un escenario más complejo. El error (Fitness) se mide como el número de elementos que el robot deja sin encontrar. Aquí presentamos únicamente un versión abreviada del enunciado del problema, los detalles se pueden consultar en [Koz92].

La gramática de este problema se especifica en la tabla 5.4. Los símbolos terminales son las acciones que puede realizar el robot: avanzar o girar (derecha o izquierda). De los tres símbolos funcionales, dos de ellos son de composición: $P2$ y $P3$, se ejecuta en orden (de izqd. a dcha.) todos sus argumentos, 2 y 3 respectivamente. El restante, $IF-FOOD-AHEAD$, es un operador condicional: si existe comida en la casilla siguiente del mapa hacia donde está orientado el robot entonces se ejecuta el primer argumento de la expresión, en caso contrario se ejecuta el segundo.

Las expresiones representan programas de control que determinan el comportamiento del robot. El robot dispone de cierta energía: E, la cual es consumida cada vez que avanza o gira. El programa de control se ejecuta de manera contínua hasta que el robot se queda sin energía (la energía no se recupera por encontrar comida). Cuanta menos energía disponga, más eficiente tiene que ser la expresión de control para permitirle encontrar todos los elementos del mapa. Plantearemos diferentes niveles de dificultad variando la energía disponible en cada escenario.

Functores (G_F)	IF-FOOD-AHEAD(2), P2(2), P3(3)
Terminales (G_T)	LEFT, RIGHT, FORWARD

Tabla 5.4: Especificación de la gramática para el problema "Artificial Ant Trail". Entre paréntesis se indica la aridad de cada símbolo funcional.

Un ejemplo de una expresión con esta gramática se muestra en la figura 5.18, la expresión simboliza el comportamiento siguiente: "Si hay comida delante avanza una casilla. Si no la hay, gira a la izquierda y avanza una casilla". El robot repetirá esta pauta de comportamiento hasta que se quede sin energía.

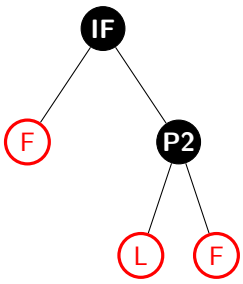


Figura 5.18: Ejemplo de una expresión utilizando la gramática del "Artificial Ant".

"Santa Fe trail" "Santa Fe" es el primer escenario propuesto por Koza, ver figura 5.19, como se puede apreciar es un mapa que tiene un tamaño de 32×32 y contiene 89 elementos a encontrar (comida). En la imagen también se puede ver marcada la casilla de inicio del robot.

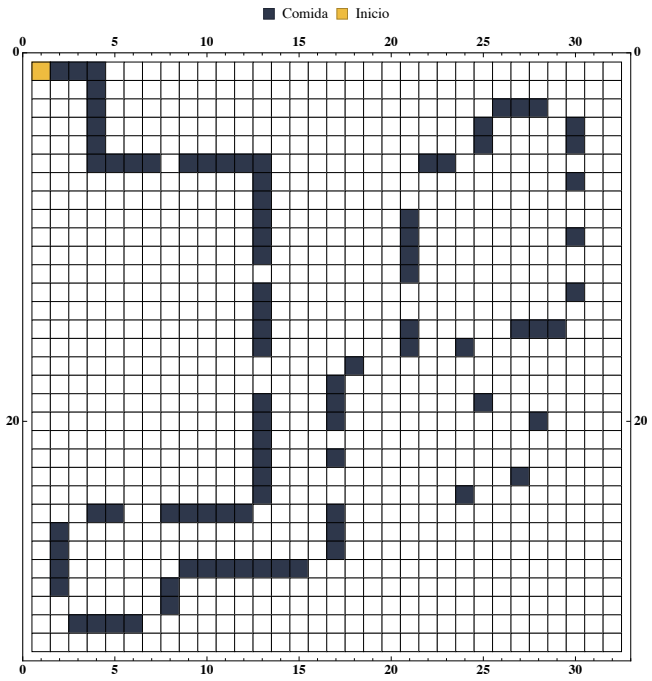
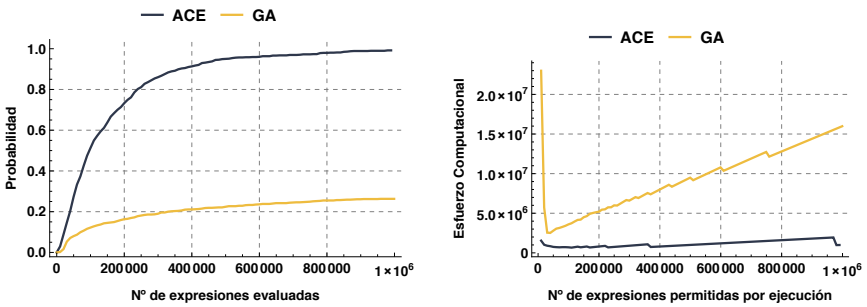


Figura 5.19: Escenario de los "Santa Fe".

Los resultados de este primer estudio, permitiendo una energía máxima de $E = 400$, se pueden ver en la figura 5.20. El rendimiento de ACE (Esfuerzo computacional mínimo $= 0{,}66 \times 10^6$) es 2,8 veces inferior al del algoritmo genético (Esfuerzo computacional mínimo $= 2{,}51748 \times 10^6$).

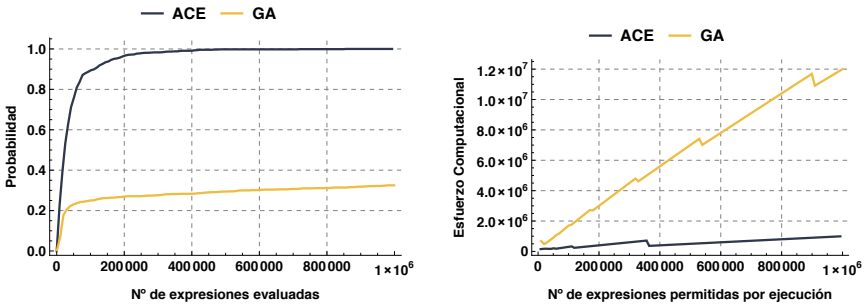


(a) Probabilidad de satisfacer el predicado.

(b) Esfuerzo computacional.

Figura 5.20: Probabilidad de encontrar satisfacer el predicado y esfuerzo computacional requerido, con una energía disponible $E = 400$.

Si aumentamos la energía disponible en el robot, $E = 600$, éste puede moverse más tiempo luego es más fácil encontrar una expresión que satisfaga el predicado. Como se puede ver en la figura 5.21, ambos algoritmos aumentan su rendimiento. El rendimiento de ACE (Esfuerzo computacional mínimo $= 142741$) sigue siendo inferior, aproximadamente 2,1 veces, al del algoritmo genético (Esfuerzo computacional mínimo $= 479520$).



(a) Probabilidad de satisfacer el predicado.

(b) Esfuerzo computacional.

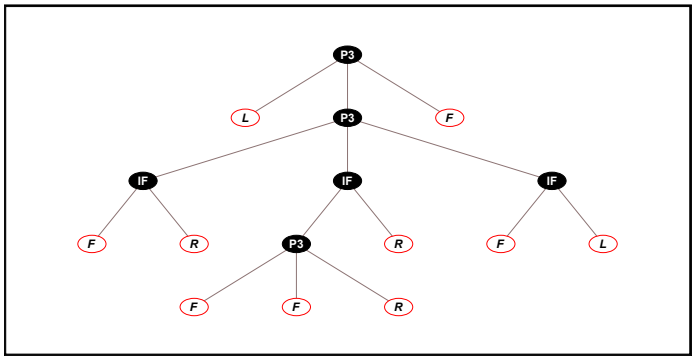
Figura 5.21: Probabilidad de encontrar satisfacer el predicado y esfuerzo computacional requerido, con una energía disponible $E = 600$.

En la tabla 5.5 se muestran el tamaño de las expresiones generadas por cada algoritmo y que satisfacen el predicado. Como se puede observar, ACE utiliza expresiones con la décima parte de nodos que un algoritmo genético. Cuantos menos nodos, menor es el efecto del "Bloat", ya que es necesario que los nodos cumplan algún papel a la hora de que la expresión satisfaga el predicado pedido. Un ejemplo de las expresiones generadas por ACE se muestra en la figura 5.22.

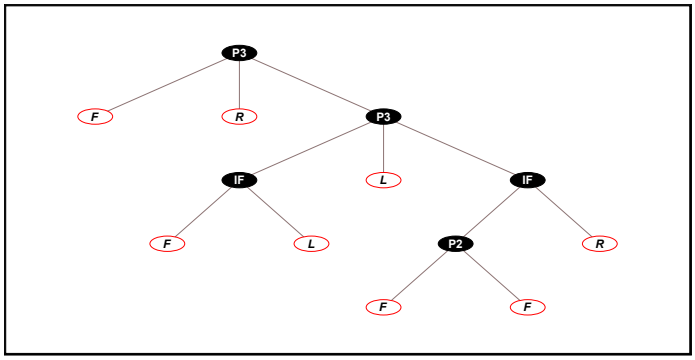
ACE	GA	
35	388	Media
31	329	Mediana
17	269	Desviación Estándar

ACE	GA	
25	221	Media
21	127	Mediana
15	256	Desviación Estándar

Tabla 5.5: Estadística descriptiva sencilla del nº de nodos de aquellas expresiones que satisfacen el predicado. La tabla superior es en el caso de $E = 400$ y la inferior cuando $E = 600$.



(a) $E = 400$.



(b) $E = 600$.

Figura 5.22: Expresión generada por ACE para la solución del "Artificial Ant" utilizando el escenario de "Santa Fe".

En este problema concreto es interesante advertir que el hecho de evaluar los árboles atómicos parece dar una ventaja significativa a ACE, ya que un enfoque similar (analizar las estructuras pequeñas para componer otras más complejas) es el estudio de programación genética que consigue los mejores resultados en este escenario [CO07].

"Los Altos trail" "Los Altos" es el segundo escenario propuesto por Koza, ver figura 5.23, como se puede apreciar es un mapa mucho más grande (100×100) que el escenario de "Santa Fe" y contiene un mayor número de elementos ($comida = 157$).

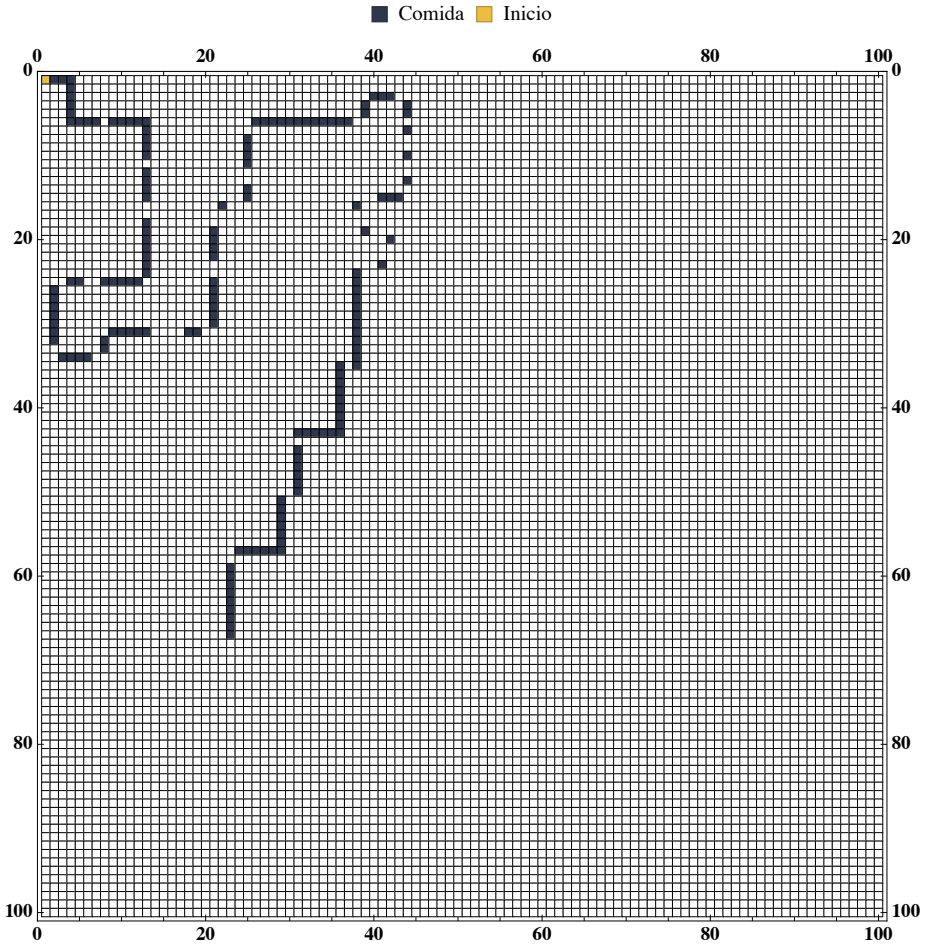
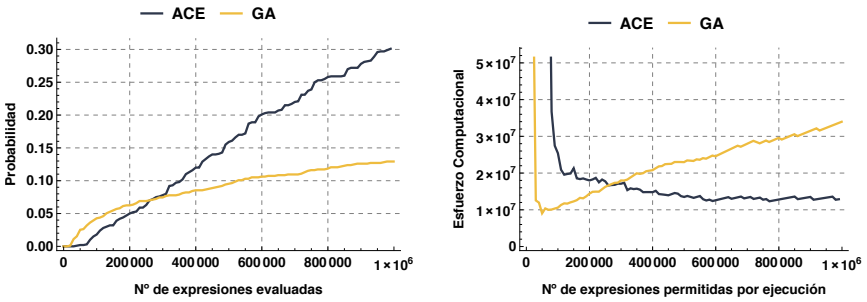


Figura 5.23: Escenario de los "Los Altos".

Las condiciones experimentales son las mismas que para el escenario anterior excepto la introducción de un símbolo funcional nuevo: $P4(4)$, para componer instrucciones con cuatro órdenes. Además se incrementa la energía disponible para el robot: $E \in \{4000, 6000\}$.

Los resultados permitiendo una energía máxima de $E = 4000$ se pueden ver en la figura 5.24. Aunque ACE tiene una mayor probabilidad de encontrar una expresión adecuada, el algoritmo genético consigue encontrar soluciones muy rápidamente, lo que se traduce en un menor esfuerzo computacional: $ACE = 1{,}12801 \times 10^7$ y $GA = 9{,}04095 \times 10^6$. ACE requiere un 36 % más de esfuerzo que el algoritmo genético para garantizar satisfacer el predicado con un 99 % de probabilidad.

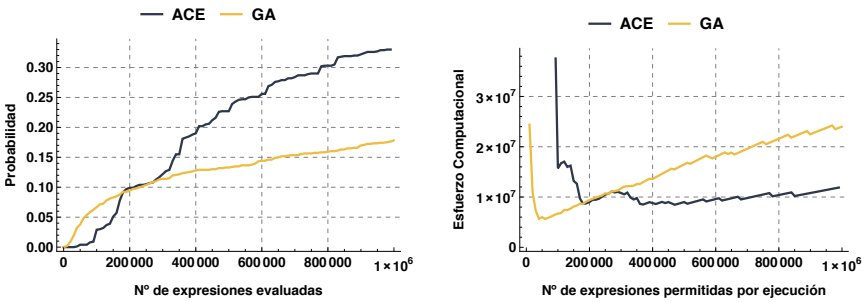


(a) Probabilidad de satisfacer el predicado.

(b) Esfuerzo computacional.

Figura 5.24: Probabilidad de encontrar satisfacer el predicado y esfuerzo computacional requerido, con una energía disponible $E = 4000$.

Si aumentamos la $E = 6000$ ocurre lo mismo, ver figura 5.25. La probabilidad de ACE es mayor, pero crece más lentamente, lo que se traduce en un mayor esfuerzo computacional $ACE = 8{,}46 \times 10^6$ y $GA = 5{,}63 \times 10^6$, 1,5 veces menos esfuerzo que ACE.



(a) Probabilidad de satisfacer el predicado.

(b) Esfuerzo computacional.

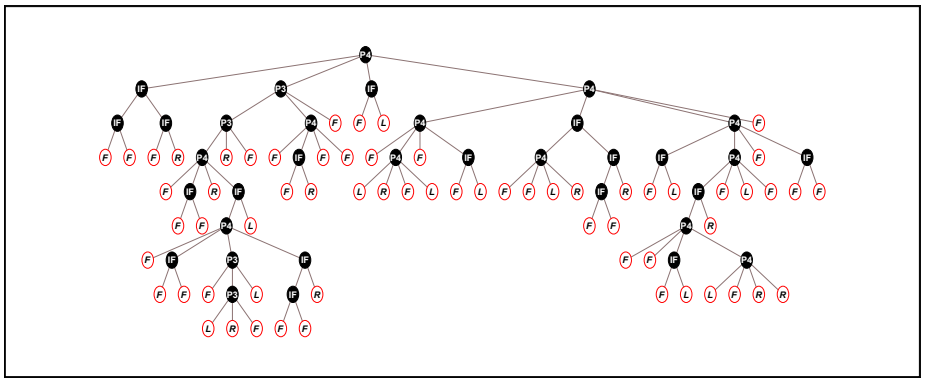
Figura 5.25: Probabilidad de encontrar satisfacer el predicado y esfuerzo computacional requerido, con una energía disponible $E = 6000$.

Si atendemos al tamaño de las expresiones generadas, las desarrolladas por ACE son muchísimo más pequeñas que las del algoritmo genético, ver tabla 5.6. Concretamente, el genético genera expresiones aproximadamente con un número de nodos 41 veces más elevado que ACE. Un ejemplo de las expresiones generadas por ACE se muestra en la figura 5.26.

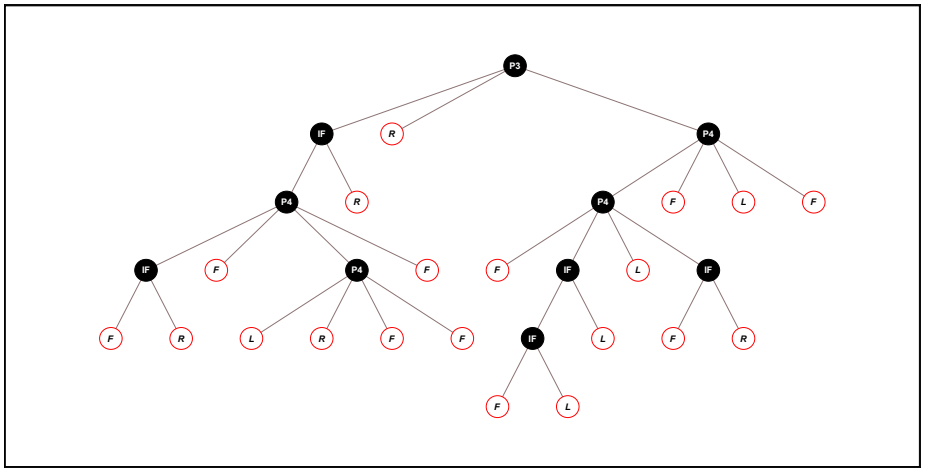
ACE	GA	
48	1921	Media
38	1651	Mediana
29	1317	Desviación Estándar

ACE	GA	
41	1964	Media
35	1738	Mediana
28	1211	Desviación Estándar

Tabla 5.6: Estadística descriptiva sencilla del nº de nodos de aquellas expresiones que satisfacen el predicado. La tabla superior es en el caso de $E = 4000$ y la inferior cuando $E = 6000$.



(a) $E = 4000$.



(b) $E = 6000$.

Figura 5.26: Expresión generada por ACE para la solución del "Artificial Ant" utilizando el escenario de "Los Altos".

El hecho de que el algoritmo genético sea capaz de encontrar soluciones evaluando menos expresiones es algo que cabría esperar, pues ACE es una meta-

heurística constructiva. ACE para alcanzar cierto tamaño en las expresiones tiene que aprender a construirlas, lo que consume un cierto número de evaluaciones. En cambio, el genético ya dispone de expresiones de varias decenas de nodos en la inicialización, hacerlas crecer no le cuesta ningún esfuerzo. Como se necesita un tamaño mínimo para que la expresión satisfaga el problema, por debajo de un cierto número de nodos no existen soluciones, el algoritmo genético alcanza este punto con muchas menos evaluaciones que ACE, lo que se traduce en un menor esfuerzo computacional.

Multiplexor

Este problema consiste en encontrar una expresión lógica que concuerde con la tabla de verdad de un multiplexor [Her98]. Un multiplexor se compone de una serie de variables de dirección y otra serie de variables de datos. La salida se corresponde con el valor del dato indicado por el valor de las variables de dirección.

Por ejemplo, la tabla de un multiplexor con una sola variable de dirección ($A0$) es la siguiente:

$A0$	$D0$	$D1$	Z
0	0	1	0
1	0	1	1

cuando $A0 = 0$ entonces $Z = D0$, y cuando $A0 = 1$, entonces $Z = D1$.

Dependiendo del número de variables de dirección podemos obtener dos configuraciones del problema: un multiplexor de 11 bits con 3 variables de dirección y un multiplexor de 6 bits, con 2 variables de dirección.

La gramática de este problema se especifica en la tabla 5.7. En este caso se especifica la gramática para un multiplexor de 11 bits (número de símbolos terminales) con 3 variables de dirección. Los símbolos terminales son las variables de datos (DX) y dirección (AX). En la búsqueda, el tipo de cada variable es desconocido, no existe distinción entre dirección y datos, para el algoritmo todos son variables. Los símbolos funcionales son funciones lógicas estándar, excepto la sentencia $IF(3)$ que es un condicional, de tal modo que si el primer argumentos es cierto, se devuelve el valor del segundo argumento. En caso de que no sea cierto el primer argumento, se devuelve el valor del tercer argumento.

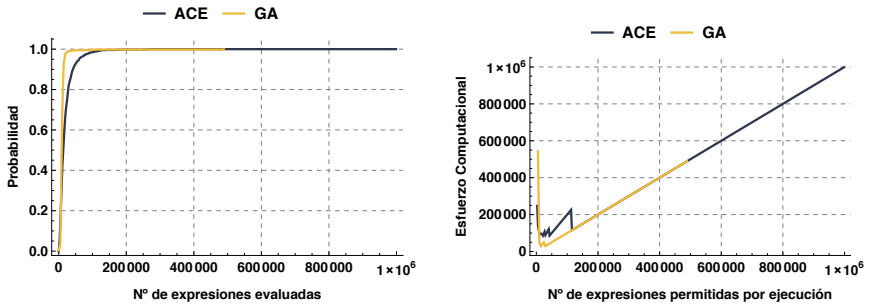
Functores (G_F)	AND(2),NOT(1),OR(2),IF(3)
Terminales (G_T)	A0,A1,A2,D0,D1,D2,D3,D4,D5,D6,D7

Tabla 5.7: Especificación de la gramática para el problema "Multiplexor". Entre paréntesis se indica la aridad de cada símbolo funcional.

Multiplexor 6 El primer estudio consiste en resolver la versión más sencilla del multiplexor: 6 bits con dos variables de dirección ($A0$, $A1$) y cuatro variables de datos ($D0, D1, D2, D3$). En este caso la tabla de verdad contiene 64 posibles entradas que la expresión lógica debe cumplir.

Los resultados se pueden ver en la figura 5.27. Ambos algoritmos no tienen ninguna dificultad en solucionar el problema. El Algoritmo Genético (Esfuerzo

computacional mínimo = 29340) es 1,92 veces más eficiente que ACE (Esfuerzo computacional mínimo = 85901).



(a) Probabilidad de satisfacer el predicado.

(b) Esfuerzo computacional.

Figura 5.27: Probabilidad de encontrar satisfacer el predicado y esfuerzo computacional requerido, para un multiplexor con 2 variables de dirección (6 bits).

Una de las expresiones generadas por ACE que satisface el predicado se muestra en la figura 5.28, como se puede apreciar apenas contiene el número mínimo de nodos (10) para satisfacer el predicado. En general, ver la tabla 5.8, las expresiones finales son 4 veces más grandes, pero en ningún caso alcanzan el tamaño de las del algoritmo genético.

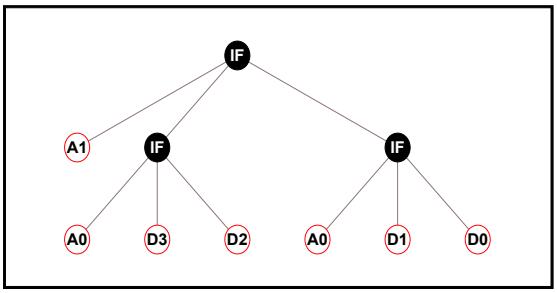


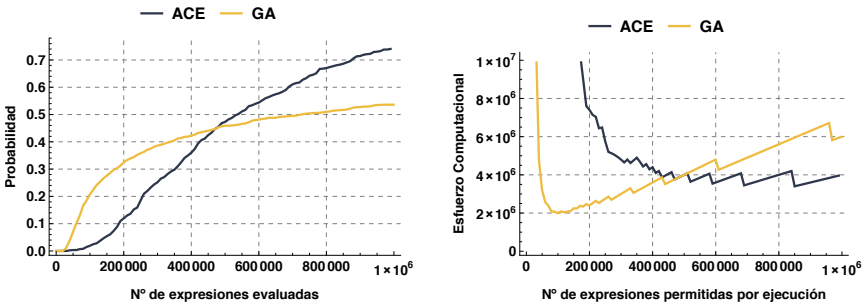
Figura 5.28: Expresión generada por ACE para la solución del multiplexor de 6 bits.

ACE	GA	
40	118	Media
36	75	Mediana
20	120	Desviación Estándar

Tabla 5.8: Estadística descriptiva sencilla del nº de nodos de aquellas expresiones que satisfacen la tabla de verdad del multiplexor de 6 bits.

Multiplexor 11 Este estudio consiste en resolver la versión más compleja del multiplexor: 11 bits con tres variables de dirección. En este caso la tabla de verdad contiene 2048 posibles entradas que la expresión lógica debe cumplir.

Los resultados se pueden ver en la figura 5.29. Como se puede ver ambos algoritmos son capaces de resolver el problema satisfactoriamente. El Algoritmo Genético (Esfuerzo computacional mínimo = $1{,}998 \times 10^6$) es 1,7 veces más eficiente que ACE (Esfuerzo computacional mínimo = $3{,}4 \times 10^6$) porque aunque ACE sea capaz de dar con una expresión correcta con mayor probabilidad, requiere un mayor número de evaluaciones para conseguirlo.



(a) Probabilidad de satisfacer el predicado.

(b) Esfuerzo computacional.

Figura 5.29: Probabilidad de encontrar satisfacer el predicado y esfuerzo computacional requerido, para un multiplexor con 3 variables de dirección (11 bits).

Una de las expresiones generadas por ACE que satisface el predicado se muestra en la figura 5.30, como se puede apreciar contiene muchos más nodos que en el caso del multiplexor de 6 bits. Aún así, de media utiliza únicamente 61 nodos, bastantes menos que los casi 1000 del algoritmo genético. La estadística de tamaño se muestra en la tabla 5.9.

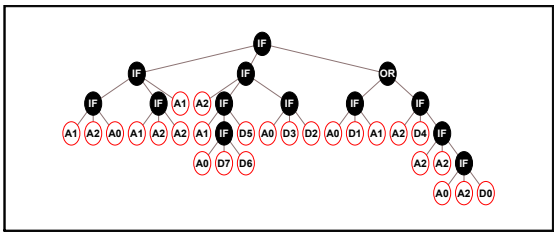


Figura 5.30: Expresión generada por ACE para la solución del multiplexor de 11 bits.

ACE	GA	
61	969	Media
53	905	Mediana
31	451	Desviación Estándar

Tabla 5.9: Estadística descriptiva sencilla del nº de nodos de aquellas expresiones que satisfacen la tabla de verdad del multiplexor de 11 bits.

Paridad impar

Este problema consiste en encontrar una expresión lógica que de como resultado 1 cuando el número de 1 de una serie de variables sea impar. Por ejemplo, la tabla de la paridad de 2 bits es la siguiente:

$D0$	$D1$	Z
0	0	0
0	1	1
1	0	1
1	1	0

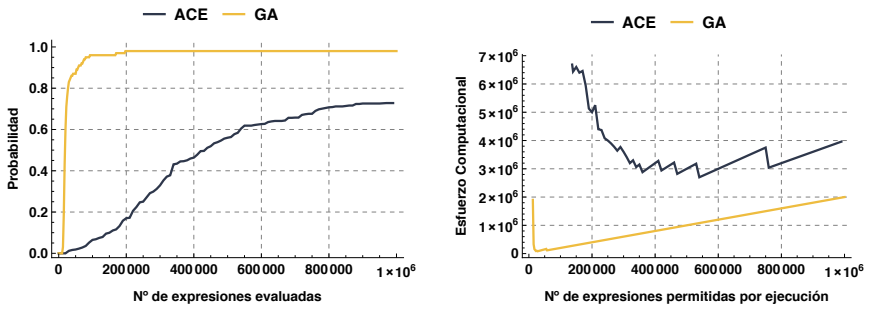
La gramática de este problema para el caso de 4 bits se especifica en la tabla 5.10.

Functores (G_F)	AND(2),NOR(1),OR(2),NAND(2),XOR(2)
Terminales (G_T)	D0,D1,D2,D3

Tabla 5.10: Especificación de la gramática para el problema "Paridad Impar". Entre paréntesis se indica la aridad de cada símbolo funcional.

En este caso nos vamos a plantear dos situaciones: a) Que los términos funcionales incluyan la función XOR o que no la incluyan. Si no incluimos la función XOR la dificultad del problema se incrementa.

Sin incluir la función XOR Los resultados se pueden ver en la figura 5.31. Como se puede ver el Algoritmo Genético (Esfuerzo computacional mínimo = 82500) es claramente más eficiente (31 veces más eficiente) que ACE (Esfuerzo computacional mínimo = $2{,}7 \times 10^6$).



(a) Probabilidad de satisfacer el predicado.

(b) Esfuerzo computacional.

Figura 5.31: Probabilidad de encontrar satisfacer el predicado y esfuerzo computacional requerido, para la paridad impar sin incluir la función XOR.

Aunque el algoritmo genético sea más eficiente, como se puede ver en la tabla 5.11, genera expresiones aproximadamente 5 veces mayores que las generadas por ACE. Aún así, ACE genera expresiones de aproximadamente 100 nodos, como la que se muestra en la figura 5.32.

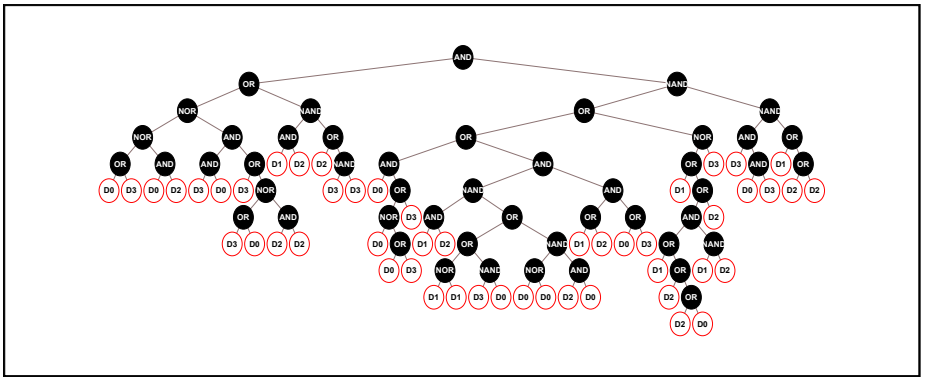


Figura 5.32: Expresión generada por ACE para la solución de la paridad impar sin incluir la función XOR.

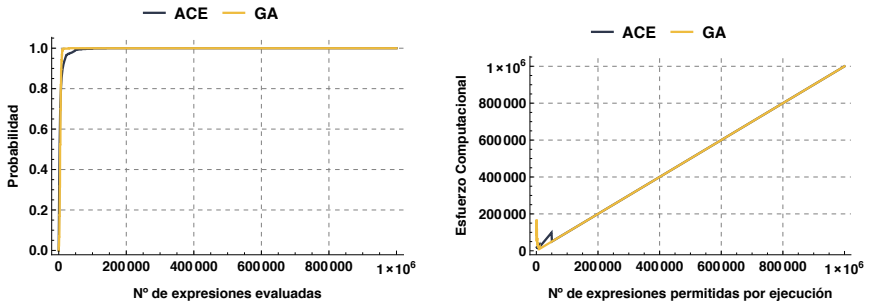
ACE	GA	
90	528	Media
87	481	Mediana
43	228	Desviación Estándar

Tabla 5.11: Estadística descriptiva sencilla del nº de nodos de aquellas expresiones que satisfacen la tabla de verdad de la paridad impar sin incluir la función XOR.

El problema de la paridad es complejo ya que se necesita construir estructuras complejas con funciones lógicas básicas, como la respuesta es un valor de lógico ($\top$, $\bot$), el valor máximo de error que obtenemos para una expresión es un 50 %. Es decir, existen infinidad de expresiones que arrojan el mismo resultado, la función objetivo no ofrece una guía clara, sino grandes planicies donde un gran conjunto de expresiones arrojan el mismo valor.

Una estrategia bastante eficaz para resolver este tipo de problemas, donde la complejidad es proporcional al número de variables, consiste en resolver el caso más sencillo. Esta solución la encapsulamos como un procedimiento auxiliar que utilizamos como elemento estructural para resolver las instancias más complejas. Para el caso de la paridad, la solución más simple (2 bits) es la función XOR. Por tanto, podemos resolver el mismo problema (paridad de 4 bits) pero incluyendo la función XOR.

Incluyendo la función XOR Los resultados se pueden ver en la figura 5.33. Como se puede apreciar al incluir la función XOR la dificultad se reduce considerablemente. El Algoritmo Genético (Esfuerzo computacional mínimo = 11160) es ligeramente más eficiente (1,5 veces más eficiente) que ACE (Esfuerzo computacional mínimo = 16876).



(a) Probabilidad de satisfacer el predicado.

(b) Esfuerzo computacional.

Figura 5.33: Probabilidad de encontrar satisfacer el predicado y esfuerzo computacional requerido, para la paridad impar incluyendo la función XOR.

Aunque ambos algoritmos al disponer de la función XOR construyen expresiones con un menor número de nodos que cuando no disponen de dicha función, como se puede ver en la tabla 5.12, el algoritmo genético requiere expresiones con un número de nodos 4 veces superior a las utilizadas por ACE. Un ejemplo de las expresiones generadas por ACE, la podemos ver en la figura 5.34.

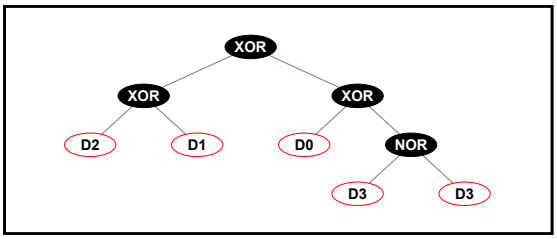


Figura 5.34: Expresión generada por ACE para la solución de la paridad impar sin incluir la función XOR.

ACE	GA	
27	97	Media
25	83	Mediana
15	61	Desviación Estándar

Tabla 5.12: Estadística descriptiva sencilla del nº de nodos de aquellas expresiones que satisfacen la tabla de verdad para la paridad impar incluyendo la función XOR.

Se puede ver como al disponer de un elemento de construcción más eficaz XOR y no tener que desarrollar estructuras complejas, la diferencia respecto al algoritmo genético se ha reducido considerablemente. Dada la naturaleza del genético, concretamente al operador de cruce, este es más eficiente conservando estructuras completas y re-utilizándolas. En cambio, ACE si quiere utilizar la misma estructura repetida en distintas partes de una expresión debe construirla. Dependiendo de la complejidad de la estructura a repetir, este proceso será más o menos difícil.

Problemas contínuos: regresiones matemáticas

Este último estudio constituye el segundo bloque de experimentos, se diferencia de los anteriores fundamentalmente en que la función de evaluación es contínua.

El problema consiste en aproximar una serie de funciones matemáticas mediante una serie de puntos generados al azar a partir de la definición general de la función. Utilizando el punto y el valor de la función en dicho punto, obtenemos una tabla para evaluar las diferentes expresiones matemáticas que genera el algoritmo. Un punto se considera evaluado correctamente cuando el error absoluto en dicho punto es inferior a 0,01. Se considera que se ha aproximado correctamente la función cuando se evalúan correctamente todos los puntos de la tabla.

Al tratarse de números reales, podemos utilizar el error cuadrático medio para evaluar una expresión. Esto cambia radicalmente frente a los problemas anteriores, porque en muchos casos una modificación de una expresión, al ser la función de evaluación discreta, no producía cambio alguno en el resultado. Ahora en cambio, al ser contínua, cualquier cambio en la expresión se ve reflejado en el valor de la función de evaluación.

La lista de funciones que vamos a evaluar ha sido tomada de [KOKG12], se muestran en la tabla 5.13.

Función	Puntos a evaluar
$F_1 = x^3 + x^2 + x$	20 puntos $\subseteq [-1, 1]$
$F_2 = x^4 + x^3 + x^2 + x$	20 puntos $\subseteq [-1, 1]$
$F_3 = x^5 + x^4 + x^3 + x^2 + x$	20 puntos $\subseteq [-1, 1]$
$F_4 = x^6 + x^5 + x^4 + x^3 + x^2 + x$	20 puntos $\subseteq [-1, 1]$
$F_5 = sin(x^2)cos(x) - 1$	20 puntos $\subseteq [-1, 1]$
$F_6 = sin(x) + sin(x + x^2)$	20 puntos $\subseteq [-1, 1]$
$F_7 = log(x^2 + 1) + log(x + 1)$	20 puntos $\subseteq [0, 2]$
$F_8 = \sqrt{x}$	20 puntos $\subseteq [0, 4]$
$F_9 = sin(x) + sin(y^2)$	100 puntos $\subseteq [-1, 1] \times [-1, 1]$
$F_{10} = 2sin(x)cos(y)$	100 puntos $\subseteq [-1, 1] \times [-1, 1]$

Tabla 5.13: Conjunto de funciones para realizar la regresión.

La gramática utilizada se muestra en la tabla 5.14, para las funciones de una sola variable únicamente se utiliza el terminal X. El símbolo $rlog$ es el logaritmo definido positivo: $rlog(0) = 0$.

Functores (G_F)	$+, -, \times, \div, sin, cos, exp, rlog$
Terminales (G_T)	X,Y

Tabla 5.14: Especificación de la gramática utilizada para las regresiones matemáticas.

La comparativa se realiza tomando los datos de [KOKG12], según estos términos, únicamente vamos a medir la probabilidad de que el algoritmo sea capaz de encontrar una expresión adecuada. Como forma de contabilizar el tiempo de búsqueda, ya no se define un número máximo de expresiones a evaluar, sino un

número máximo de nodos. Se permite evaluar hasta un total de 15×10^6 nodos por ejecución. Esta forma de contabilizar la búsqueda es más justa porque el esfuerzo de evaluar un nodo es el mismo para todos los algoritmos.

La configuración de ACE es la misma que hemos utilizado en los experimentos anteriores (sección 5.2). Para obtener la estadística, cada función se realizan 1000 ejecuciones independientes.

Para la comparación tomamos los resultados de [KOKG12] que es un estudio del 2012 donde se compara un algoritmo de colonia de abejas (ABCP) con un conjunto de algoritmos genéticos que utilizan diferentes operadores de cruce desarrollados para programación genética: standard crossover (SC), no same mate (NSM) selection, semantics aware crossover (SAC), context aware crossover (CAC), soft brood selection (SBS), and semantic similarity-based crossover (SSC). Para cada algoritmo, cada función se resuelve de manera independiente 100 veces. Cada ejecución permite evaluar hasta un total de 15×10^6 nodos.

Los resultados podemos verlos en la figura 5.35 y una estadística descriptiva en la tabla 5.15, como se puede comprobar ACE supera con mucho el rendimiento del resto de algoritmos. Hay que señalar que la comparativa se ha realizado frente a un estudio del 2012 que incluye técnicas del estado del arte actual.

Algoritmo	Media	Mediana	Desviación Std.	Maximo	Mínimo
ABCP	0.466	0.435	0.266966	0.89	0.12
SC	0.212	0.19	0.142891	0.48	0.04
NSM	0.216	0.18	0.158899	0.48	0.04
SAC2	0.184	0.15	0.158198	0.53	0.04
SAC5	0.181	0.145	0.154593	0.53	0.01
CAC4	0.161	0.14	0.0999389	0.35	0.03
SBS42	0.304	0.28	0.136561	0.51	0.1
SBS44	0.305	0.335	0.111878	0.43	0.09
SSC12	0.412	0.425	0.186536	0.67	0.12
SSC16	0.4	0.415	0.177326	0.67	0.11
ACE	0.8075	0.865	0.22	0.999	0.334

Tabla 5.15: Estadística descriptiva de la probabilidad de encontrar la expresión valida para cualquier función de la indicadas previamente.

Aparte de la comparativa, podemos ver cuál es el porcentaje de éxito que ha tenido ACE aproximando cada función. Para cada función, las expresiones han sido simplificadas para facilitar la legibilidad. Los resultados se muestran en la tabla 5.16. Como se puede ver todas las expresiones han sido aproximadas correctamente con una probabilidad mínima del 20 %.

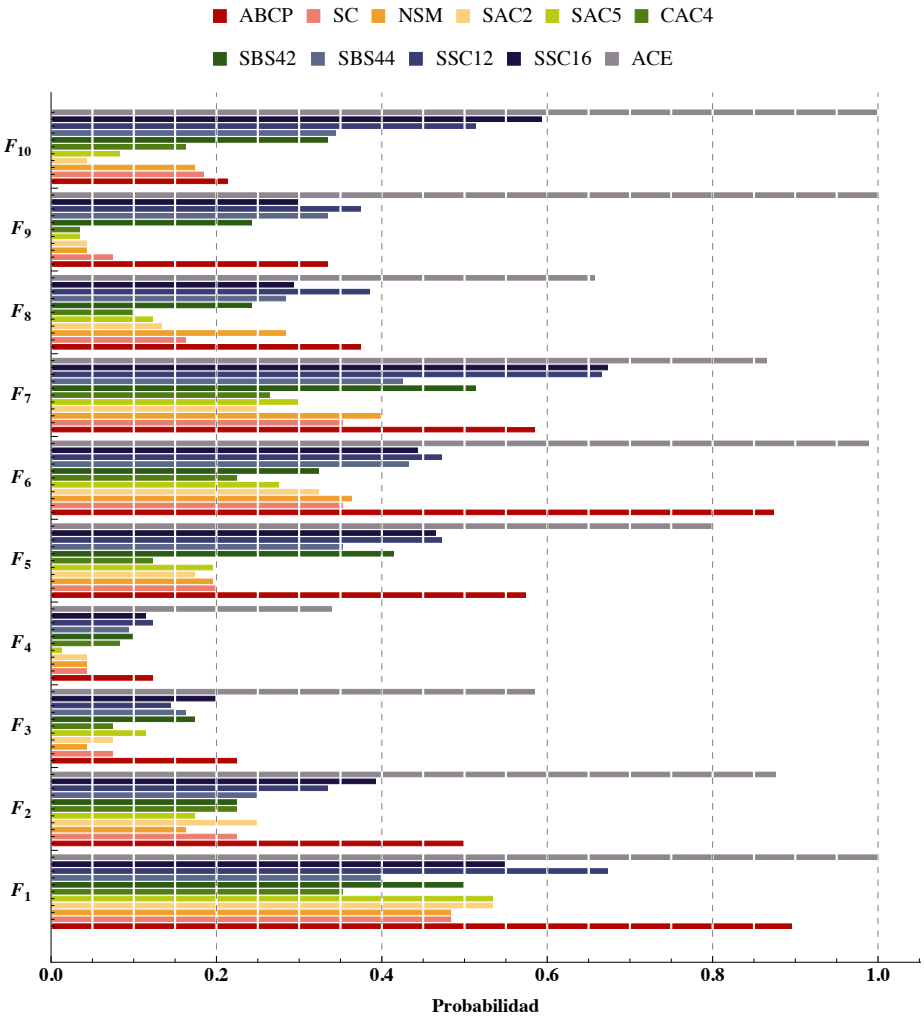


Figura 5.35: Probabilidad de encontrar la expresión que ajuste todos los puntos generados.

Función original	Expresión aproximada	
$F_1 = x^3 + x^2 + x$	$x^3 + x^2 + x$	60 %
$F_2 = x^4 + x^3 + x^2 + x$	$x^4 + x^3 + x^2 + x$	28 %
$F_3 = x^5 + x^4 + x^3 + x^2 + x$	$x^5 + x^4 + x^3 + x^2 + x$	28 %
$F_4 = x^6 + x^5 + x^4 + x^3 + x^2 + x$	$x^6 + x^5 + x^4 + x^3 + x^2 + x$	21 %
$F_5 = sin(x^2)cos(x) - 1$	$sin(x^2)cos(x) - 1$	33 %
$F_6 = sin(x) + sin(x + x^2)$	$sin(x) + sin(x + x^2)$	67 %
$F_7 = log(x^2 + 1) + log(x + 1)$	$log(x^2 + 1) + log(x + 1)$	29 %
$F_8 = \sqrt{x}$	$\sqrt{x}$	57 %
$F_9 = sin(x) + sin(y^2)$	$sin(x) + sin(y^2)$	81 %
$F_{10} = 2sin(x)cos(y)$	$2sin(x)cos(y)$	58 %

Tabla 5.16: Conjunto de de las expresiones más comunes encontradas por ACE para aproximar cada función. Para cada expresión se indicada el porcentaje de casos en los que se ha dado como solución final la expresión indicada.

Este ejemplo sirve para ilustrar cómo el desarrollo automático de expresiones conduce a resultados correctos que probablemente nunca se nos ocurrirían. Por ejemplo, para la función F_8, una de las formas aproximada de la función se muestra en la figura 5.36, si la escribimos matemáticamente resulta la expresión:

$$e^{\frac{\log(x)}{\frac{x}{x} - \frac{x}{(x-x)-x}}}$$

la cual simplificada es la función $\sqrt{x}$. Pero como en la gramática no se especifica la función "'potencia" ni constantes numéricas como el 1 y el 2, el algoritmo tiene que obtenerlas de otra manera. Por ejemplo la potencia x^y la escribe como $e^{\log(x)*y}$. Como no están escritas como habituamos, nos resultan expresiones "raras".

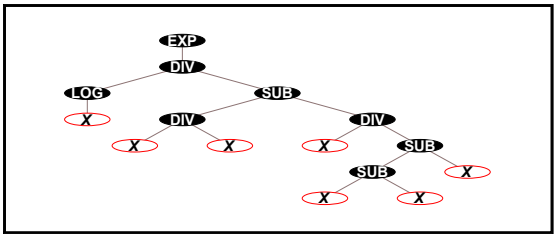


Figura 5.36: Expresión generada por ACE para aproximar la función $\sqrt{x}$.

Discusión

Si nos centramos en los resultados del primer bloque de experimentos ("Artificial Ant", Multiplexor y paridad impar), hay un dato interesante: en general ACE aumenta la probabilidad de satisfacer el predicado según aumenta el número de evaluaciones permitidas. Es indicativo de que la búsqueda no se atasca. Esto coincide con los resultados obtenidos en la comparativa del TSP (sección 5.1) que es un problema radicalmente diferente.

En cambio, el algoritmo genético aunque muestra una probabilidad alta en los primeros estadios de la búsqueda, ésta no aumenta tan significativamente según

el número de evaluaciones. De cara al esfuerzo computacional, como las ejecuciones de un algoritmo son sucesos independientes, conviene ejecutar un algoritmo genético muchas veces y con pocas evaluaciones para satisfacer el predicado. Por el contrario, ACE debe procesar un mayor número de evaluaciones para que una ejecución aislada alcance una probabilidad mínima de satisfacer el predicado.

Esto quiere decir que en algunos problemas, como los que incluyen funciones lógicas, el algoritmo genético parece ser más eficiente que ACE. Pero hay que considerar que las expresiones generadas son de tamaños significativamente distintos. Nosotros hemos respetado el planteamiento original de Koza por mantener una medida de referencia utilizando el algoritmo genético. Pero para que el esfuerzo computacional fuese comparable habría que medir las evaluaciones en nodos, no en expresiones completas. Ya que si la desviación entre el tamaño de las expresiones es significativa, esto afecta al esfuerzo: no es lo mismo evaluar 700 nodos que 100.

El último experimento de la regresión matemática es significativo de esta consideración. Como la evaluación tiene en cuenta el número de nodos, ACE es capaz de probar un número amplio de diferentes expresiones, en vez de probar unas pocas complejas. Lo que se traduce en una mejor exploración y un mayor rendimiento comparado con el resto de algoritmos.

Otro dato interesante que aportan estos experimentos es la capacidad de ACE para generar expresiones donde el efecto del "Bloat" es mucho menos significativo de partida. Es cierto que las expresiones generadas contienen términos redundantes. Pero como son expresiones muchos más pequeñas, la redundancia respecto a un algoritmo genético también es mucho menor. Lo que incrementa la calidad de la soluciones generadas sin necesidad de utilizar técnicas auxiliares de post-procesado.

A la vista de los resultados obtenidos ACE puede constituir un buen método de partida para la solución de problemas de programación genética. Obviamente el estado del arte de los algoritmos genéticos esta más avanzado, pero el problema del "Bloat" es mucho menos significativo de partida, lo cual constituye lo más atractivo de estos resultados.

5.3. Planificación de maniobras para barcos

Este último trabajo experimental se diferencia de los anteriores en tanto que ya no interesa el estudio del algoritmo en sí, sino su aplicación para resolver un problema práctico: la planificación y optimización de maniobras viables para un barco autónomo. El objetivo consiste en obtener la trayectoria descrita por un barco entre dos puntos empleando el mínimo tiempo posible. Se parte de un estado inicial con velocidad y orientación arbitraria, y se trata de alcanzar el punto de destino con unos valores predeterminados para la velocidad y la orientación. La dinámica especifica del barco impone las correspondientes restricciones a su maniobrabilidad.

Para conseguir este objetivo, se utiliza un modelo simplificado de la dinámica de un barco, considerando únicamente tres grados de libertad (avance, guiñada y deriva). La propulsión se ha modelado utilizando un sistema de waterjets ajustable, que juega también el papel de timón. La velocidad y el rumbo se ajustan utilizando un controlador PID clásico que estabiliza el rumbo y la velocidad. Una ilustración básica de las coordenadas empleadas se puede ver en la figura 5.37,

la parametrización del modelo se indica en la tabla 5.17. Para más detalles de la descripción del modelo se puede consultar [EJGS11, EJGS12, EJG09, EJG10b].

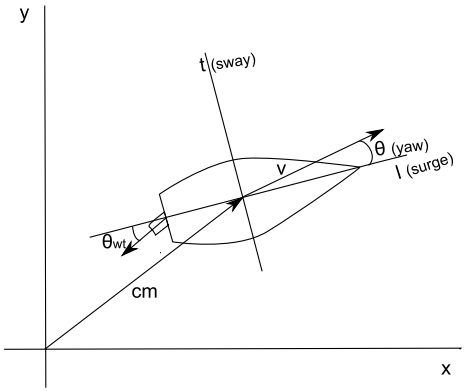


Figura 5.37: Esquema general de las coordenadas empleadas.

	Parámetro	símbolo	valor
1	Eslora	l_s	$4m$
2	Masa	m_b	$100kg$
3	Fuerza máxima Waterjets	F_{max}	$2500N$
4	Ángulo máximo de giro del waterjet	θ_{wtmax}	$45grados$
5	Componente líneal de la resistencia al avance	μ_{ll}	$7N.s/m$
8	Componente cuadrática de la resistencia al avance	μ_{lc}	$10N.s^2/m^2$
9	Componente líneal de la resistencia al desplazamiento lateral	μ_{tl}	$1000N.s/m$
10	Componente cuadrática de la resistencia al desplazamiento lateral	μ_{tc}	$10000N.s^2/m^2$
11	Coeficiente líneal de las masas añadidas en la dirección l	r_{mal}	$0{,}5kg.s/m$
12	Coeficiente líneal de la masas añadidas en la dirección t	r_{mat}	$10Kg.s/m$
13	Momento de inercia	I_b	$kg.m$
14	Coeficiente líneal del momento de inercia añadido	r_{Iba}	1 kg.m.s/rad
15	Coeficiente líneal de la resistencia al giro	$\mu_a l$	$10N.m.s/rad$
16	Coeficiente cuadrático de la resistencia al giro	$\mu_a c$	$100N.m.s^2/rad^2$

Tabla 5.17: Valores de los parámetros que componen el modelo

Descripción de un escenario de maniobras

El escenario de maniobras se especifica mediante dos estados: inicial y final. Un estado queda descrito en función de las siguientes variables:

$$[x(t), y(t), v_l(t), \theta(t)]$$

donde x, y indica la posición del barco en coordenadas cartesianas, v_l es el valor de la velocidad en la dirección longitudinal y θ el ángulo de apuntamiento.

Para considerar que se ha alcanzado el estado final ($[x^f, y^f, v_l^f, \theta^f]$), se trabaja

con un cierto grado de tolerancia:

$$|x(t) - x^f| \leq 5m$$
$$|y(t) - y^f| \leq 5m$$
$$|v_l(t) - v_l^f| \leq 1m/s$$
$$|\theta(t) - \theta^f| \leq 7{,}5^o$$

estos valores se han ajustado en función de las características del barco.

La maniobra queda completamente determinada dando las posiciones, orientaciones y velocidades iniciales y finales del barco. La transición entre el estado inicial y el final debe hacerse respetando las restricciones diferenciales impuestas por la dinámica del barco.

En nuestro caso lo que vamos a buscar es una secuencias de órdenes para el controlador PID de a bordo de tal modo que su utilización permita realizar la maniobra pedida. La solución no se especifica como puntos a seguir, sino como órdenes a realizar:

$$U_v = [U_v^1, U_v^2, \ldots U_v^n]$$
$$U_\theta = [U_\theta^1, U_\theta^2, \ldots U_\theta^n]$$

donde U_v representa una secuencia de consignas para el controlador de la velocidad, y U_θ representa una secuencia para el controlador de rumbo.

Dado un escenario podrían establecerse dos objetivos, el primero sería encontrar trayectorias posibles que permitan completar la maniobra, el segundo sería encontrar de entre ellas una que sea óptima en algún sentido. En el presente trabajo se aborda la búsqueda de trayectorias que minimicen el tiempo empleado en recorrerlas, es decir el objetivo es minimizar la función $\Delta t = t_f - t_i$.

Desde el punto de vista de la búsqueda de las trayectorias, el problema pierde sentido si la distancia que separa la posición inicial de la final del barco es excesivamente grande. En este caso bastaría con descomponer el movimiento del barco, en una maniobra de partida –una trayectoria en línea recta hacia la posición objetivo– y una maniobra final de aproximación hacia el objetivo. El concepto de maniobra cobra realmente sentido cuando se desarrolla en distancias cortas, porque la maniobra exige cambios de rumbo de cierta complejidad. Estos casos serán los que compondrán los escenarios de trabajo.

A continuación se describen algunos ejemplos de escenarios de maniobra. Los escenario se describen en base a los valores de los estados inicial y final de cada maniobra.

- Virado en redondo sobre la dirección de avance: se trata de hacer un viraje de 180 grados sobre la propia trayectoria del barco. Un ejemplo posible de esta maniobra sería,

$$E_i = [0m, 0m, 0m/s, 90^o] \rightharpoonup E_f = [0m, 20m, 4m/s, 270^o]$$

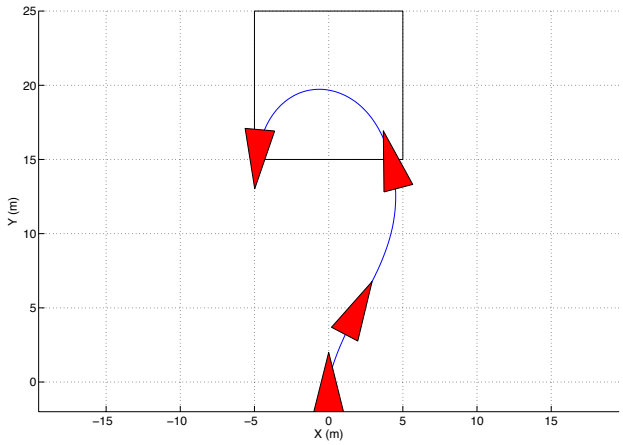


Figura 5.38: Ejemplo de maniobras: virado en redondo sobre la dirección de avance.

- Desplazamiento lateral: el barco debe alcanzar una posición a babor o estribor de su posición inicial, con el mismo rumbo con que inició la maniobra. Por ejemplo,

$$E_i = [0m, 0m, 0m/s, 90^o] \rightharpoonup E_f = [20m, 0m, 15m/s, 90^o]$$

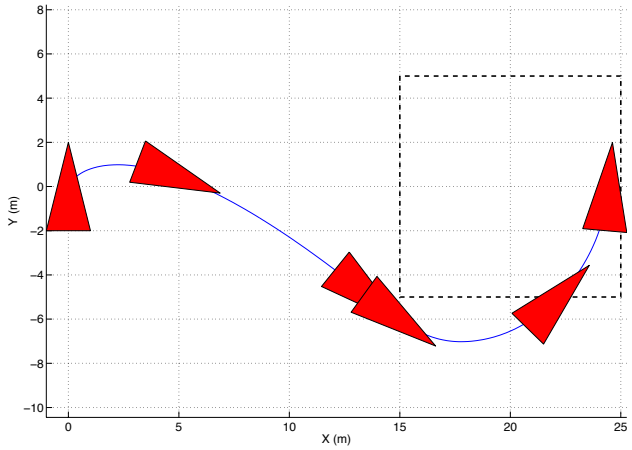


Figura 5.39: Ejemplo de maniobras: desplazamiento lateral.

Las maniobras descritas tienen el interés de exigir grandes cambios de rumbo en distancias relativamente cortas, a la vez que ponen de manifiesto las restricciones al movimiento del barco. Aunque en estos ejemplos no se indique, se considera la presencia de obstáculos en el escenario de maniobras.

Representación

Hay casos de planificación de maniobras/rutas con algoritmos heurísticos que son escenarios donde no existen –o son mínimas– las restricciones al movimiento del vehículo. Por ejemplo, se asume que un vehículo puede detenerse en el acto. Esto no quiere decir que este tipo de asunción sea incorrecta, sino que nuestro escenario es otro. El objetivo es planificar considerando las restricciones impuestas por la dinámica del vehículo, de este modo se garantiza que la maniobra planificada puede ser realizada por el barco modelado. Si no tenemos en cuenta las restricciones dinámicas, entonces cuando el barco real quiera realizar la maniobra, precisará de un complejo cálculo que adapte la trayectoria planificada al tipo de movimiento que puede realizar, pudiendo incluso llegar a ser imposible. Por ejemplo una maniobra que exija detenerse prácticamente en el acto, sin considerar la inercia, puede ser inviable dependiendo de la velocidad del barco.

Para satisfacer las restricciones diferenciales lo que hacemos es *utilizar el modelo diferencial de la dinámica como función de transición de estados*. De este modo cuando el algoritmo calcule el resultado de aplicar una acción, establecer unos valores de consigna para el rumbo y la velocidad, el nuevo estado viene dado por las características dinámicas del barco.

El enfoque utilizado en ACE no exige conocer el espacio de estados previamente, sino que simplemente la propia exploración es la construcción de dicho espacio. De tal modo que el algoritmo comienza sin saber nada acerca de como "mover" el barco. A partir de maniobras "de prueba", el algoritmo irá obteniendo el conocimiento necesario para guiar al barco hasta el estado final.

El modelo diferencial del barco es contínuo, mientras que ACE es un algoritmo que trabaja en espacio de estados discretos. Por tanto la representación del problema está basada en una discretización.

Acciones: son pares de órdenes de consigna. Las consignas de velocidad las discretizamos en intervalos de $1/ms$ hasta la velocidad máxima del modelo ($14/m/s$), y el rumbo en intervalos de 1^o:

$$[u_v, u_\theta]$$

donde $u_v \in \{1m/s, 2/m/s, \ldots 14m/s\}$ y $u_\theta \in \{1^o, 2^o, \ldots 360^o\}$.

Estados: proyectamos la posición real del barco sobre un espacio discreto, utilizando un mallado que permite agrupar los estados contínuos. Se asigna una malla al espacio real, donde cada celda tiene un tamaño proporcional al error que se permite ($10m \times 10m$).

Este mallado se construye según las siguientes ecuaciones, donde x, y son las posición real y X, Y la posición discreta.

$$X = 10 * sign(x) * |round(\frac{x}{10})| \tag{5.4}$$

$$Y = 10 * sign(y) * |round(\frac{y}{10})| \tag{5.5}$$

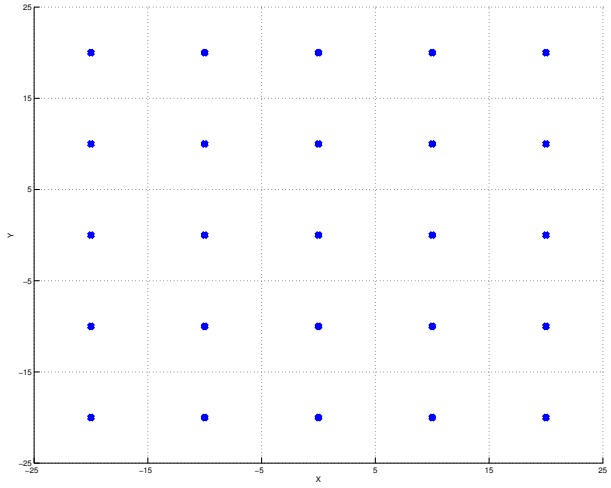


Figura 5.40: Discretización del espacio, empleando un mallado de celdas de 10x10m

En la figura 5.41 podemos ver la proporción entre el tamaño del mallado y el barco. El tamaño de celda (10m) se ha escogido considerando la eslora del barco (4m) para permitir el movimiento en su interior.

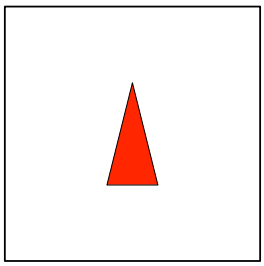


Figura 5.41: Proporción entre el tamaño del mallado y el barco.

El objetivo fundamental del mallado es definir los estados discretos que va a utilizar ACE para almacenar la información en la tabla de feromona. En la tabla almacenamos asociaciones de consignas y rumbo con posiciones discretas: las celdas. De este modo aunque los estados del barco sean del espacio real, al proyectarlos sobre el mallado podemos saber cuáles son las órdenes de consigna que debemos aplicar en función de la celda donde se encuentra el barco.

Función de transición: la transición entre estados viene dada por la integración numérica del modelo del barco. Para ilustrar el proceso vamos a considerar que el barco se encuentra inicialmente en el estado $\mathbf{x}(0) = [x_0, y_0, \theta_0, v_{l0}, v_{t0}, \omega_{b0}]$. Si proyectamos la posición inicial x_0, y_0, obtenemos una primera celda inicial: $\mathbf{x}(0) \mapsto L_{ij}$. A continuación seleccionamos un par de valores de consigna

asociados a la celda actual: $X_1 = [r_{hd}(L_{ij}), r_{vl}(L_{ij})]$. Por último integramos el modelo diferencial del barco hasta que se alcance el borde de la celda actual:

$$\mathbf{x}(T_1) = \int_0^{T_1} \dot{\mathbf{x}}(t)dt$$

donde T_1 representa el tiempo que tarda el barco en alcanzar el borde de la celda actual.

El barco se encuentra localizado en una nueva celda: $\mathbf{x}(T_1) \mapsto L_{kl}$, la cual es adyacente a L_{ij}. Se seleccionan un par de nuevos valores de consigna ($X_2 = [r_{hd}(L_{kl}), r_{vl}(L_{kl})]$) y se vuelve a integrar el modelo hasta que se alcanza el borde de la celda L_{kl}.

$$\mathbf{x}(T_2) = \int_{T_1}^{T_2} \dot{\mathbf{x}}(t)dt$$

El barco se mueve en un espacio contínuo gobernado por un par de valores de consigna, los cuales son válidos mientras el barco permanezca en la celda actual. Cada vez que se cambia de celda, se obtienen unos nuevos valores de consigna, utilizando el mallado actual para determinar cuales utilizar.

Todo el proceso se puede considerar como una especie de sistema híbrido: el barco se mueve en un espacio contínuo gobernado por eventos discretos: cruzar la frontera de una celda. En la figura 5.42 se muestra una imagen de la relación entre las trayectorias contínuas y el mallado discreto.

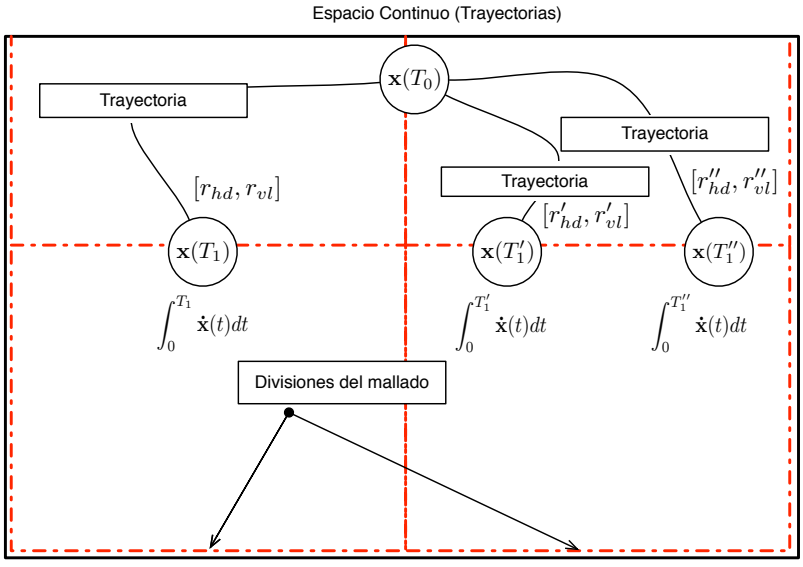


Figura 5.42: Diagrama tomado de [EJGS12] que muestra la relación entre las trayectorias contínuas del barco y el mallado discreto.

Heurística

Para que ACE sea capaz de resolver una maniobra de manera eficiente es imprescindible que cuente con una heurística razonable. La heurística no tiene que ser perfecta, puesto que el algoritmo es capaz de adquirir conocimiento sobre el problema y mejorar el rendimiento que se obtendría buscando únicamente con la heurística. Pero sí que debe servir para mejorar el rendimiento de la búsqueda.

En ACE la heurística se define como distribuciones de probabilidad sobre el conjunto de acciones. Como en este caso una acción es un par de velocidad y rumbo, necesitaremos definir dos tipos de heurística.

Velocidad la heurística de la velocidad la definimos como una distribución normal discretizada, centrada en el valor de la velocidad especificado como requisito en el estado final. En caso de que el estado final no imponga ninguna velocidad concreta, centramos la distribución en el valor máximo de la velocidad. La figura 5.43 ilustra estas distribuciones.

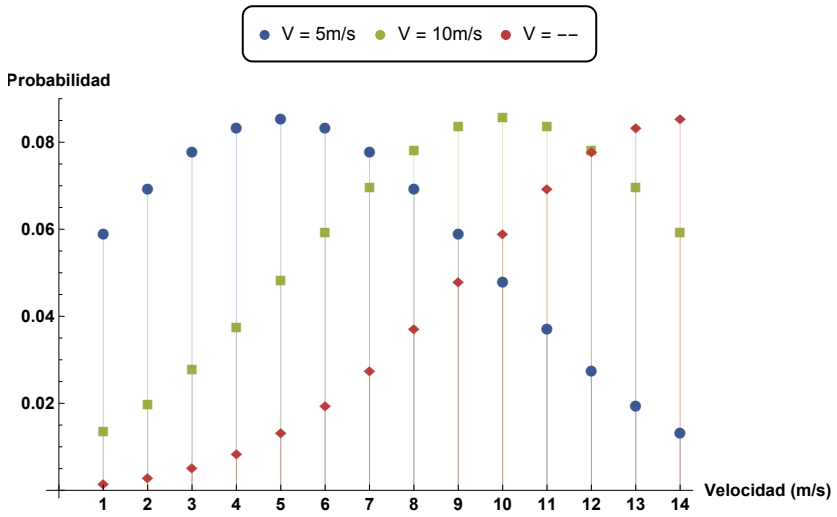


Figura 5.43: Distribuciones de probabilidad para la heurística de la velocidad para los casos donde la velocidad final está restringida y cuando está libre (——).

Rumbo la heurística del rumbo supone definir un procedimiento más complejo ya que en cierto modo debe cumplir alguna restricción del movimiento del barco y tratar con la presencia de obstáculos. Por ejemplo debe tener en cuenta que el barco no se puede mover hacia atrás, que un obstáculo se debe bordear pero yendo hacia el objetivo, etc ...

El mallado en principio es infinito, es decir no se restringe el movimiento del barco. La heurística también está asociada al mallado, pero únicamente se calcula para una zona determinada. El mallado donde se calcula la heurística consiste en un cuadrado en cuyo centro se encuentra la celda objetivo que se quiere alcanzar y en algún borde se encuentra la celda inicial. Fuera de los límites del cuadrado, si se quiere hacer uso de la heurística, entonces la posición del barco se proyecta sobre la frontera del mallado para obtener un valor de la heurística.

Para calcular la heurística vamos a utilizar un autómata celular [Wol02]. La idea es aprovechar las propiedades de auto-organización del autómata para configurar la heurística. Definimos una serie de autómata fijos, que no cambian de estado, en unas posiciones claves del mallado. Estos autómatas fijos constituyen el estado inicial del autómata celular. Poco a poco, la información contenida en los autómatas fijos se propaga por toda la red de autómatas, auto-organizándose en función de esos primeros autómatas fijos.

Los autómatas los definimos como vectores, puesto que deben indicar un rumbo. Esto nos permite definir el estado de un autómata como la media de los autómatas adyacentes utilizando un vecindario de 8 casillas.

La idea básica es muy sencilla: intuitivamente en ciertas posiciones, como puedan ser las esquinas o las posiciones adyacentes a la casilla objetivo, un obstáculo, etc... No nos cuesta mucho esfuerzo definir como debería moverse el barco en esa posición. Por ejemplo, en una esquina el rumbo debería ser hacia el centro del mallado que es donde se encuentra el objetivo. Si nosotros suministramos la información necesaria en forma de autómatas, la capacidad de auto-organización de la red nos calcula cómo debe moverse el barco en el resto de casillas sin necesidad de cálculos complejos.

Se puede ver que en realidad estamos definiendo un patrón, lo único es que en vez de definirlo explícitamente por todo el mallado, únicamente definimos unos pocos puntos (autómatas) y dejamos que la auto-organización complete el resto del patrón.

Obstáculos: son un caso especial ya que no son autómatas (no tienen estado) sino celdas que dependiendo de la casilla adyacente indican un valor de rumbo u otro. Es decir, la contribución de un obstáculo al estado de un autómata es relativa a la posición del autómata frente a dicho obstáculo. La figura 5.44 ilustra como contribuye un obstáculo a sus casillas adyacentes, como se puede ver, se definen unas direcciones que permiten circunnavegar el obstáculo en dirección al objetivo. Cuando los obstáculos adquieran estructuras más complejas el propio autómata se auto-organizará para sortearlos.

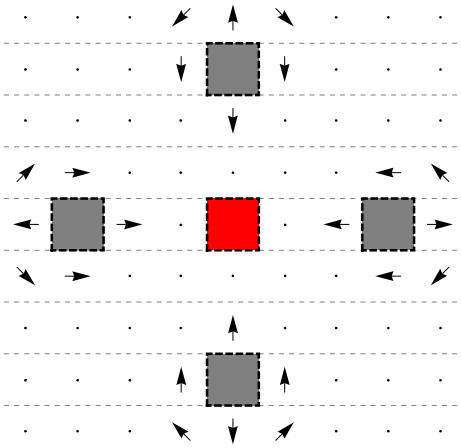


Figura 5.44: Contribución de un obstáculo a los autómatas adyacentes en función de la posición del autómata y el objetivo, marcado en rojo.

La figura 5.45 muestra un ejemplo de la evolución completa del autómata:

- El estado inicial *A* únicamente se compone de autómata fijos, dispuestos en las esquinas y alrededor de la celda objetivo. Estos autómatas tiene la misión de configurar la red. Los autómatas de los bordes mantienen el flujo hacia el objetivo. Los autómatas adyacentes al objetivo se encargan de dar forma al flujo según sea el ángulo con el que queremos alcanzar el objetivo.

- En la iteración *B*, se puede ver como los autómatas propagan su estado a los vecinos. Existe ya una pequeña auto-configuración sobre todo en las cercanías del objetivo. Los autómatas de los bordes, al estar más aislados, la única fuente de orden es el autómata fijo, y todos adquieren su mismo estado.

- En el paso *C* se puede ver como la red va tomando forma sobre todo en aquellas celdas donde el autómata tiene diversas influencias: objetivo, obstáculo, etc...

- Finalmente en *D* se puede ver la configuración final del autómata, desde la zona superior del mallado nos guía en dirección al objetivo prácticamente en línea recta, mientras que en la zona inferior nos envía bordeando los obstáculos.

La red de autómatas se utiliza para definir una distribución de probabilidad para el rumbo en cada celda, centrada en el estado del autómata asociado con dicha celda.

Por ejemplo, si un autómata tiene un ángulo de 180^o, para la celda que ocupa dicho autómata, construimos una distribución de probabilidad centrada en 180. La distribución que construimos la definimos de forma triangular para evitar que se tomen rumbos contrarios a la dirección sugerida por el autómata. En la figura 5.46 se puede ver como queda la distribución final de probabilidad para elegir un rumbo, suponiendo que un autómata tiene un ángulo de 180^o.

Hay que indicar que las heurísticas se definen mediante dos procedimientos, lo que permite generalizar su cálculo para cualquier escenario. Es decir, para cada escenario no se genera un autómata ad-hoc, sino que el autómata se auto-organiza en función del escenario. Estrictamente hablando es siempre el mismo, puesto que funciona con las mismas reglas, únicamente cambia su estado inicial, a partir del cual él se configura.

Con ambas heurísticas, ACE es capaz de planificar maniobras tanto en mar abierto como en presencia de obstáculos, sin necesidad de evaluar miles de trayectorias lo cual sería computacionalmente costoso. Como veremos en los experimentos una vez que ACE dispone de unas heurísticas adecuadas, es capaz de resolver diferentes maniobras con buena precisión y en un tiempo de cómputo razonable.

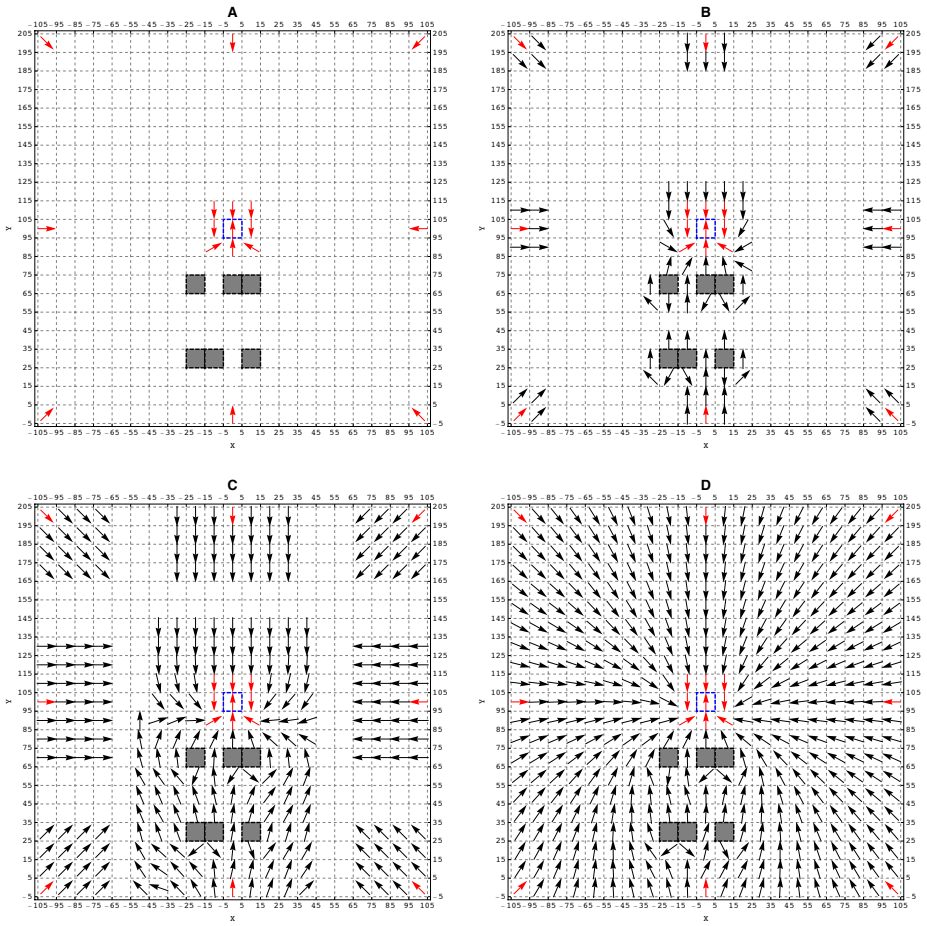


Figura 5.45: Evolución completa del autómata celular desde el estado inicial *A* hasta el final *D*. Las flechas rojas indican autómatas fijos y la celda objetivo está marcada en azul. Las celdas grises constituyen obstáculos.

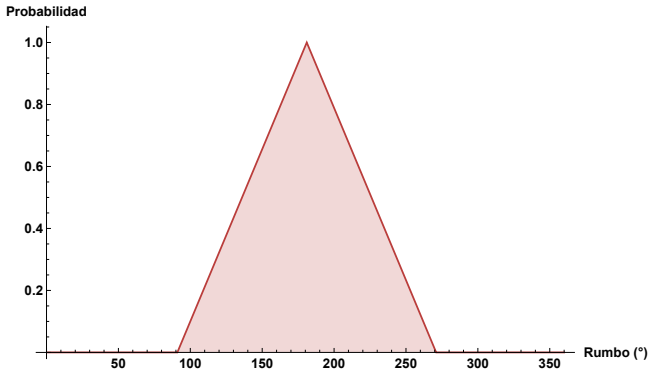


Figura 5.46: Distribuciones de probabilidad para la heurística del rumbo suponiendo un autómata con un ángulo de 180º

.

Experimentos

A continuación, presentamos y discutimos una selección de maniobras planificadas aplicando ACE a diferentes escenarios. Hemos estudiado dos tipos de posibles maniobras: a) el movimiento sólo está limitado por las restricciones dinámicas, *mar abierto*, y b) maniobras en presencia de obstáculos.

A menos que se indique lo contrario, las condiciones iniciales son siempre las mismas para todas las maniobras:

$$[x_0, y_0, \theta_0, v_{l0}] = [0m, 0m, 90^o, 0m/s]$$

Dado que las condiciones iniciales son siempre las mismas, una maniobra sólo se especifica mediante el estado final, que se describe de la siguiente manera: $[x_\tau, y_\tau, \theta_\tau, v_{l\tau}]$. En algunos casos la velocidad final puede no estar restringida, indicándose con el símbolo $--$.

Los parámetros del algoritmo son los que se indican a continuación:

Parámetro	ACE
γ_1	0,99
γ_2	0,5
γ_3	∞

Para cada escenario se realiza un conjunto de 100 experimentos de búsqueda. Un solo experimento ejecuta el algoritmo hasta que se estudian 2,000 trayectorias para realizar la maniobra.

Los resultados los presentaremos de una manera gráfica[6]: una primera figura muestra todas las trayectorias encontradas en el conjunto de las 100 búsquedas. Se utiliza un degradado de color para representar la comparación del tiempo, el gradiente va desde el rojo (trayectorias más lentas) al azul (trayectorias más rápidas). En una segunda figura se muestra la mejor solución obtenida entre todas las descubiertas, mostrando sus trayectorias, las consignas de rumbo y velocidad y la evolución del rumbo y velocidad del barco según las celdas que ha ido atravesando.

Un colección más amplia de maniobras estudiadas se puede consultar en [EJGS11, EJGS12, EJG09, EJG10b].

Maniobras en mar abierto

Escenario 1

Este escenario consiste en una navegación en línea recta de $100m$. Como la solución en este caso es conocida es posible comparar la solución del algoritmo con el óptimo: rumbo 90^o y velocidad $14m/s$.

Las condiciones finales son las siguientes:

$$[x_\tau = 0, y_\tau = 100, \theta_\tau = 90^o, v_{l\tau} = --]$$

La figura 5.47 muestra el conjunto de trayectorias encontradas para realizar la maniobra, como se puede ver, excepto alguna que se desvía un poco, la mayor de las trayectorias consisten en una navegación en línea recta.

[6] El análisis estadístico puede consultarse en [EJGS12].

En la figura 5.48 se puede ver la trayectoria encontrada que menos tiempo consume. Como se aprecia en las consignas de rumbo y velocidad, la trayectoria es muy cercana al óptimo ($\pi/2{,}14m/s$), exceptuando las últimas consignas que se utilizan para enderezar el rumbo y alcanzar el objetivo con una aptitud de 90^o.

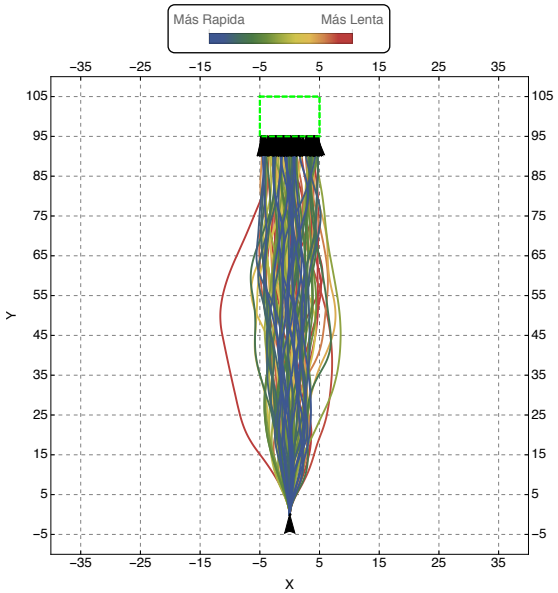


Figura 5.47: Mejores trayectorias encontradas en cada uno de los experimentos.

.

Típicamente un valor de consigna es un valor que se desea alcance el sistema, siendo el controlador responsable de que esa consigna se cumpla. En nuestro caso las consignas que el algoritmo genera no necesariamente deben alcanzarse por el barco. Como se puede apreciar en la figura 5.48 (sub-figura de rumbo), el algoritmo marca un valor alejado de $\pi/2$ para forzar al controlador PID a girar el barco. El algoritmo mantiene esa consigna hasta que el barco alcanza unos 90^o, en ese momento le indica al PID que mantenga el rumbo estableciendo una consigna de 90^o.

Aunque en los resultados pudiese parecer que se genera mucho error, ya que ninguna de las trayectorias son líneas rectas perfectas, sino que presentan oscilaciones. La trayectoria óptima de este escenario requiere un tiempo de 8,69s. El tiempo medio de las mejores trayectorias encontradas está en torno a los 8,88s, un error del 2,07 %. La mejor trayectoria encontrada consume 8,75s, lo que da un error del 0,67 %. Es decir, aunque visualmente parezca un error apreciable, en tiempo el error cometido es bastante aceptable, ya que estamos hablando del orden de décimas de segundo.

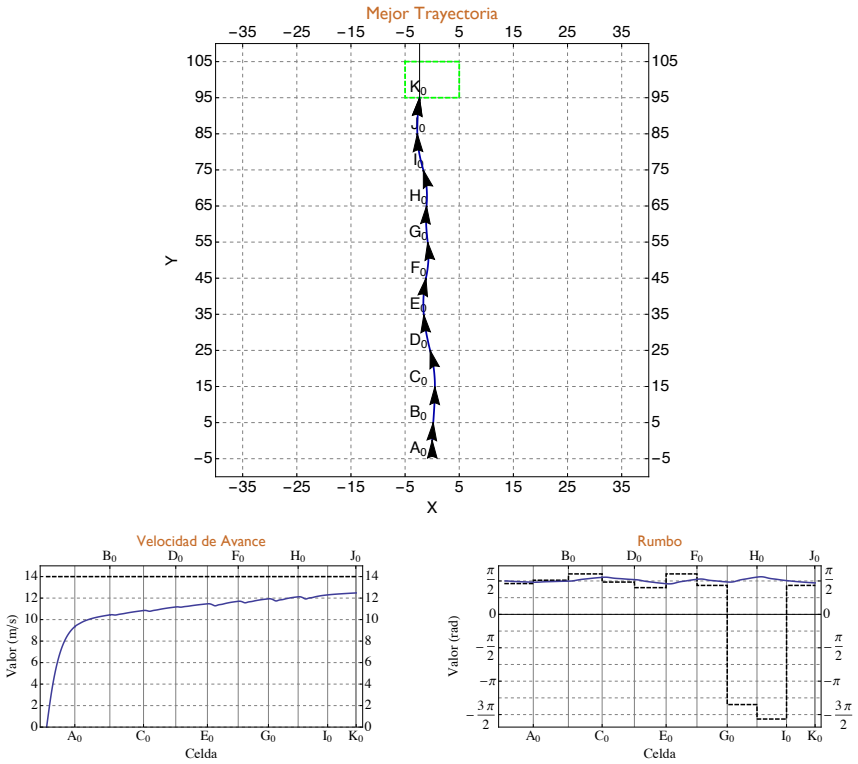


Figura 5.48: Mejor trayectoria encontrada, las sub-figuras muestran las consignas de rumbo y velocidad que describen la trayectoria: la línea azul contínua muestra la evolución del barco y la línea discontínua negra muestra los valores de consigna.

.

Escenario 2

Este escenario consiste en un desplazamiento lateral de $20m$. Las condiciones finales son las siguientes:

$$[x_\tau = 0, y_\tau = 20, \theta_\tau = 90^o, v_{l\tau} = --]$$

La figura 5.49 muestra el conjunto de trayectorias encontradas para realizar la maniobra. Un desplazamiento lateral se puede realizar de dos maneras: girando por el exterior de los dos puntos (inicial y final), o maniobrando por el interior trazando una especie de "s". Como se ve en los resultado la maniobra por el interior es bastante más rápida y la gran mayoría de soluciones se realizan maniobrando en "s".

En la figura 5.50 se puede ver la trayectoria encontrada que menos tiempo consume.

En general, cuando se quiere determinar una conducta o comportamiento, la toma de decisiones requiere especificar la secuencia de acciones y la duración de cada acción. Si bien es cierto que en muchos casos la duración de una acción viene dada por el propio contexto del problema. Por ejemplo, la planificación de un itinerario de un viaje podría ser: visitar A, visitar B, etc... Obviamente hasta que

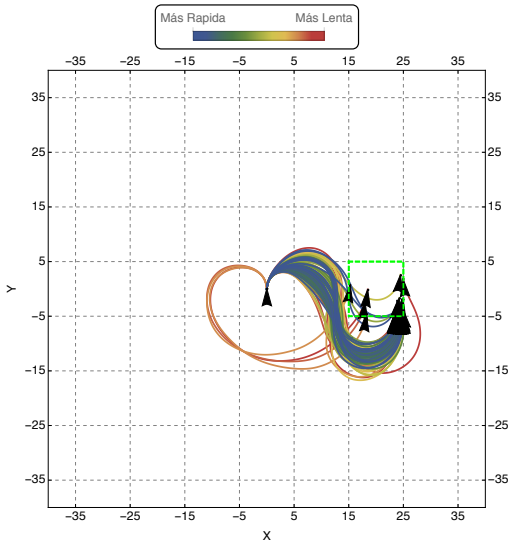


Figura 5.49: Mejores trayectorias encontradas en cada uno de los experimentos.

.

no llegamos a A, la acción "visitar A" sigue activa. En este caso la planificación, el itinerario, no requiere más especificación que una lista de puntos a donde llegar.

En el caso del barco hay que determinar la duración de un valor de consigna, porque la trayectoria varía según el tiempo que dicha consigna esté activa. Podemos denominar *distribución de la toma de decisiones* a la distribución temporal de los distintos valores que componen una trayectoria. Es decir, la duración temporal de un cierto valor de consigna.

Como hemos descrito en los apartados anteriores, se utilizan los límites de las celdas de cuadrícula para encontrar cuándo y dónde cambiar los valores de consigna. De este modo la distribución en la toma de decisiones está definida por eventos. La propia distribución surge del proceso de búsqueda, ya que no sabemos de antemano cuándo y dónde la trayectoria del ASV cruzará el límite de una celda. Una consecuencia obvia es que las decisiones no se distribuyen de manera uniforme, como se puede observar en este caso para la celda D_0 (figura 5.50).

Este tipo de políticas de decisión, basada en eventos, flexibiliza el conjunto de soluciones que se pueden obtener. Al no existir un criterio fijo para la distribución de decisiones, ésta surge en función de las acciones que se realizan, al ser los eventos una consecuencia de las acciones realizadas. Pero los eventos no quedan definidos únicamente por una acción, sino que dependen también del comportamiento del sistema. De este modo, es posible vincular la distribución de la toma de decisiones a la propia dinámica del sistema.

Es decir, la toma de decisiones no depende exclusivamente del mallado, la discretización, sino que el comportamiento del barco tiene una influencia notable en la distribución de la toma de decisiones. Una trayectoria estará en parte definida por la propia dinámica del barco, por cómo es capaz de moverse y cruzar las celdas.

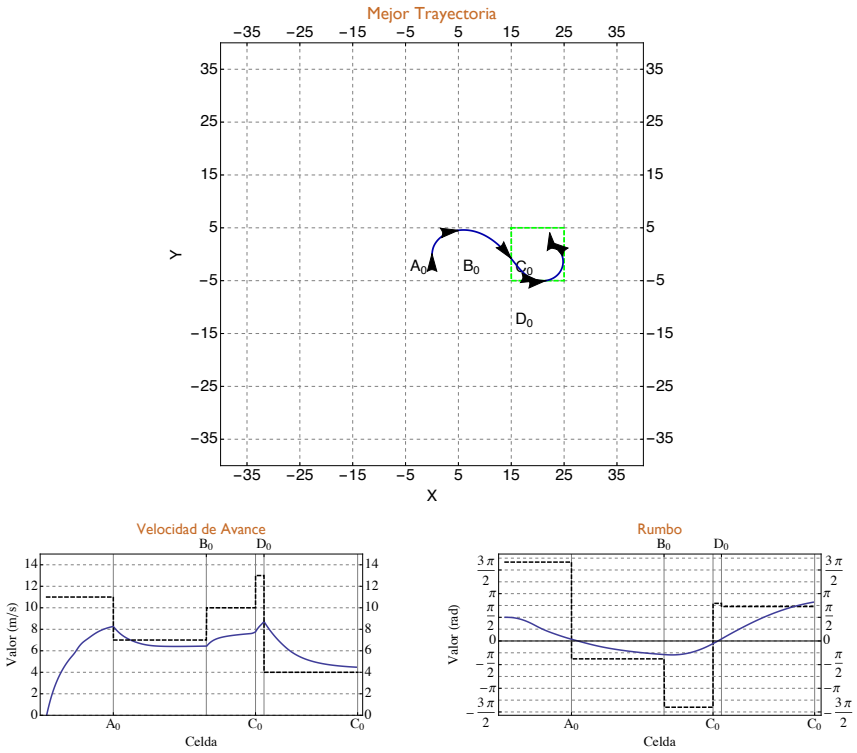


Figura 5.50: Mejor trayectoria encontrada, las sub-figuras muestran las consignas de rumbo y velocidad que describen la trayectoria: la línea azul contínua muestra la evolución del barco y la línea discontínua negra muestra los valores de consigna.

.

Maniobras en presencia de obstáculos

Escenario 1

Este escenario consiste en una navegación entre obstáculos. Las condiciones finales son las siguientes:

$$[x_\tau = 0, y_\tau = 100, \theta_\tau = 90^o, v_{l\tau} = unrestricted]$$

La figura 5.51 muestra el conjunto de trayectorias encontradas para realizar la maniobra. Lo interesante de este escenario es que existen dos barreras de obstáculos, se pueden sortear ambas por el exterior, lo que da lugar a las trayectorias más lentas ya que se tiene que recorrer un mayor espacio. Igualmente se puede atravesar la primera barrera y sortear la segunda, lo que mejora respecto a sortear ambas barreras. Por último se pueden atravesar ambas, lo que da lugar a las trayectorias más rápidas, ya que es casi una navegación en línea recta.

La mayoría de las trayectorias encontradas discurren por entre las dos barreras, lo cual indica la capacidad del algoritmo para explorar posibles soluciones diferentes. Aunque parezca trivial, no es fácil atravesar las dos barreras y cumplir con las condiciones finales de la maniobra. Muchas pruebas chocan contra los obstáculos, otras, aún consiguiendo superar los obstáculos, no satisfacen los

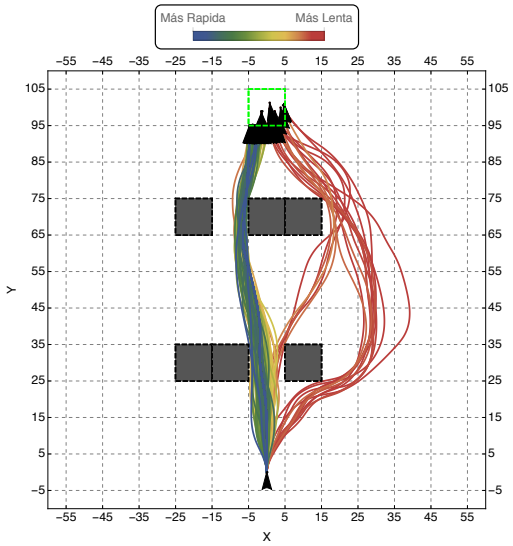


Figura 5.51: Mejores trayectorias encontradas en cada uno de los experimentos.

.

objetivos finales de la maniobra.... Cuando se descubre una primera trayectoria, como la información se almacena en la tabla de feromona, las siguientes búsquedas tienen una tasa de éxito mucho más elevada, puesto que ya existe información *precisa* de como sortear los obstáculos y cumplir los requisitos finales.

En la figura 5.52 se puede ver la mejor trayectoria encontrada, nótese como se trata prácticamente de una navegación en línea recta, con un desplazamiento suave hacia la izquierda para sortear la segunda barrera de obstáculos y una corrección final para alcanzar la celda objetivo con una aptitud de 90^o.

Este escenario sirve muy bien para ilustrar las capacidades del algoritmo. En la figura 5.53 se muestra una evolución de la búsqueda de trayectorias. Lo más interesante son las búsquedas de los foragers marcadas en rojo, porque revelan el patrón de soluciones que está generando el algoritmo en base a la información contenida en la tabla de feromona. Los patrollers van al "azar" siguiendo la heurística, por tanto es mucho más difícil apreciar el uso de la feromona en su patrón de búsqueda.

La primera solución se obtiene por el flanco izquierdo de las barreras, siguiendo la trayectoria que está repitiendo el forager. Como la heurística recomienda otro tipo de búsqueda, los patrollers tienden a salir hacia la dirección opuesta: centro-derecha. En la segunda sub-figura, se puede ver como hay patrollers que buscan por ambos flancos: izquierda, siguiendo la información de la feromona, y derecha, utilizando la heurística. Como encontrar una trayectoria que bordee los obstáculos es sencillo, rápidamente se encuentra una alternativa mejor y la búsqueda se desplaza hacia la derecha.

En la tercera sub-figura ya se ha encontrado una ruta interior. Lo interesante es advertir como existen foragers utilizando la trayectoria antigua (derecha) y otros la nueva (interior-centro). Al ser un algoritmo asíncrono, cuando se encontró la trayectoria interior ésta no se perdió, ya que la información no se "machacó" por el resto de las hormigas que utilizaban trayectorias por la derecha. Sino que

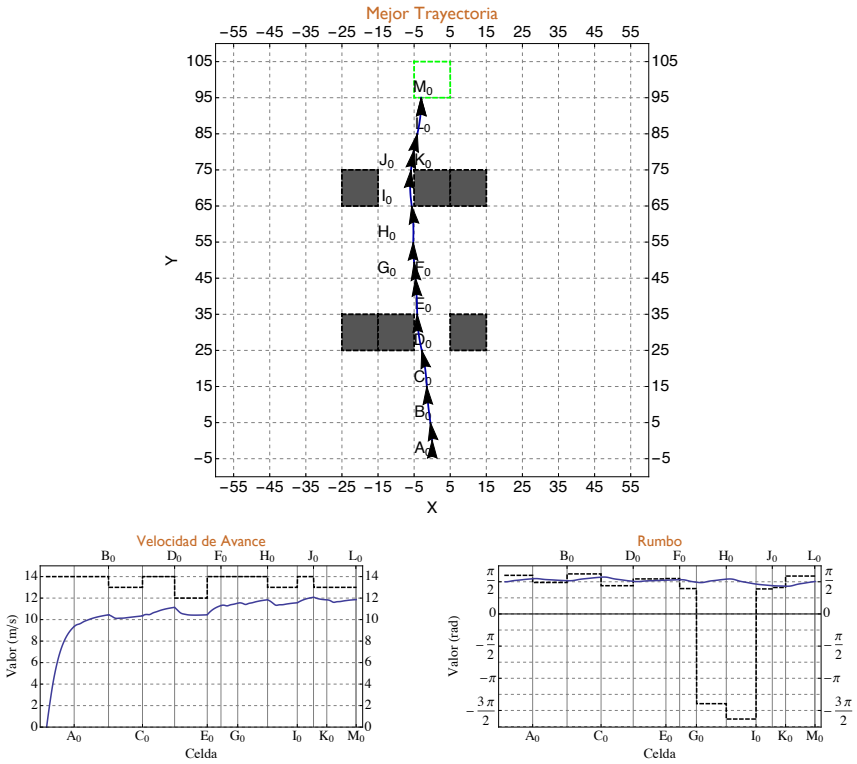


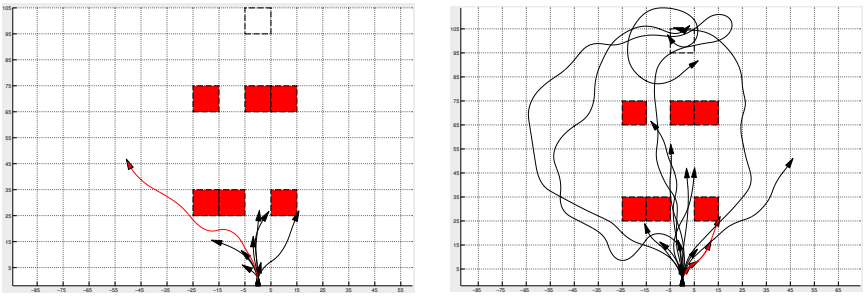
Figura 5.52: Mejor trayectoria encontrada, las sub-figuras muestran las consignas de rumbo y velocidad que describen la trayectoria: la línea azul contínua muestra la evolución del barco y la línea discontínua negra muestra los valores de consigna.

.

permaneció el tiempo suficiente en la tabla para que los foragers la utilizasen y fuesen por el interior.

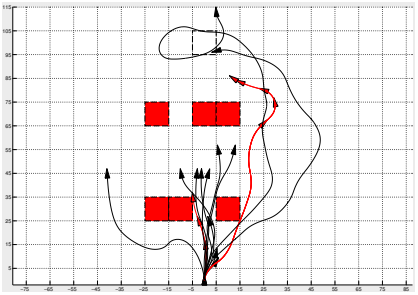
Como la información está distribuida entre los agentes: foragers en el interior y foragers por la derecha, se pueden explorar ambos tipos de trayectorias a la vez, sin necesidad de que la búsqueda se incline hacia un patrón rápidamente. Con el tiempo, aquel patrón de búsqueda que permita obtener significativamente mejores soluciones se hará predominante.

En la última sub-figura se puede apreciar cómo se converge hacia trayectorias por el interior: no existen foragers que busquen por otros lados. También es interesante advertir cómo la población es reducida para evitar dispersar la búsqueda antes de "asegurar" la información actual.

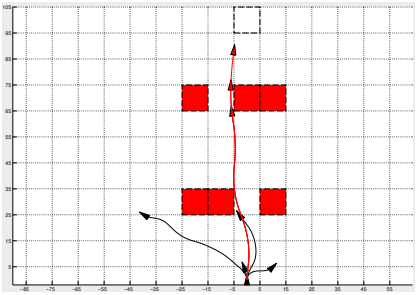


(a) Primeras soluciones.

(b) Búsqueda de trayectorias por la derecha.



(c) Búsqueda de trayectorias por el interior.



(d) Estado final.

Figura 5.53: Evolución de la búsqueda de trayectorias. Las trayectorias rojas indican la búsqueda llevada a cabo por los foragers y las negras las llevadas a cabo por patrollers.

Escenario 2

Este escenario representa una posible maniobra de aproximación a un muelle. La maniobra comienza con el barco navegando paralelo al muelle y debe finalizar con el barco situado en el muelle con una velocidad final de $1m/s$.

Las condiciones iniciales de navegación son:

$$[x_0 = 0, y_0 = 0, \theta_0 = 88^o,$$
$$v_{l0} = 12{,}3m/s, v_{t0} = 0m/s, \omega_{b0} = 0{,}3rad/s]$$

Las condiciones finales son:

$$[x_\tau = 50, y_\tau = 0, \theta_\tau = 90^o, v_{l\tau} = 1m/s]$$

La figura 5.54 muestra el conjunto de trayectorias encontradas para realizar la maniobra. Como se puede observar, debido a la inercia, el barco no comienza a girar inmediatamente sino que tiene un desplazamiento hacia delante que luego se corrige a lo largo de la maniobra.

En la figura 5.55 se puede ver la trayectoria encontrada que menos tiempo consume. Como se aprecia en las consignas de rumbo, la maniobra se compone de un viraje a derechas, una navegación en línea recta, y un viraje a izquierdas para situarse paralelo al muelle. Las consignas de velocidad son igualmente interesantes, se puede apreciar como el frenado se comienza a realizar desde la casilla G_0, ya que no existe posibilidad de un frenado inmediato, la velocidad se ha de

reducir con antelación para aproximarse al muelle lentamente.

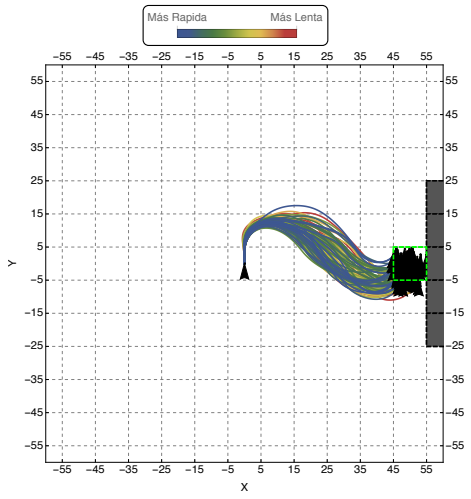


Figura 5.54: Mejores trayectorias encontradas en cada uno de los experimentos.

.

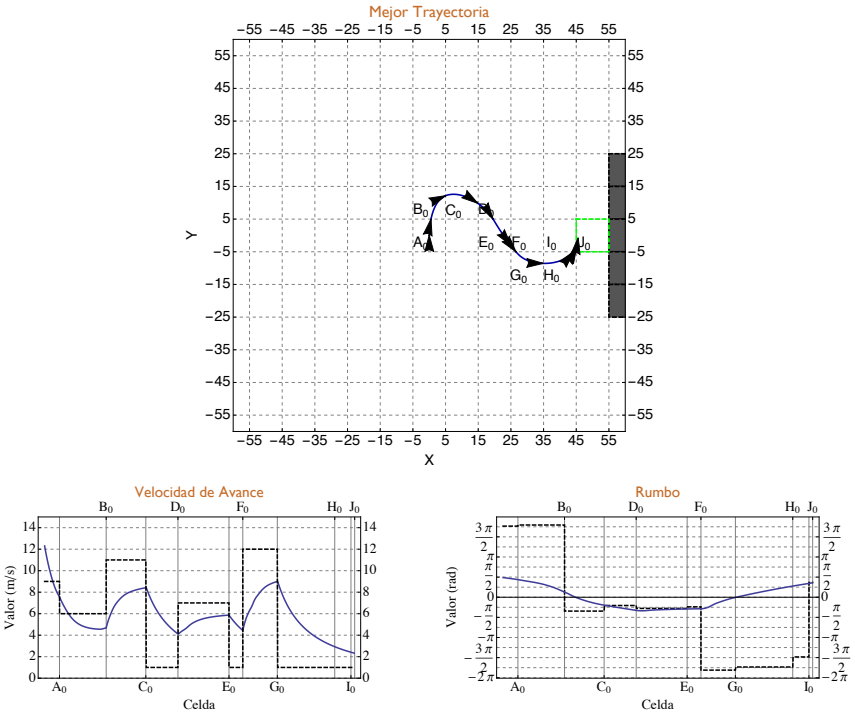


Figura 5.55: Mejor trayectoria encontrada, las sub-figuras muestran las consignas de rumbo y velocidad que describen la trayectoria: la línea azul contínua muestra la evolución del barco y la línea discontínua negra muestra los valores de consigna.

.

Escenario 3

Este escenario representa un maniobra complicada en el interior de algún tipo de estructura, por ejemplo un puerto. Tiene una pequeña entrada, una barrera en frente de dicha entrada y el estado final requiere realizar un giro de 180^o en un espacio reducido. Además la velocidad final está restringida para incrementar la complejidad.

Las condiciones finales son:

$$[x_\tau = 0, y_\tau = 100, \theta_\tau = 270^o, v_{l\tau} = 1m/s]$$

La figura 5.56 muestra el conjunto de trayectorias encontradas para realizar la maniobra. Lo interesante de este escenario es que existen trayectorias simétricas, el algoritmo es capaz de aproximar ambas. Obviamente algunas ejecuciones convergen hacia un lado u otro, pero en el cómputo global no se aprecia ningún sesgo.

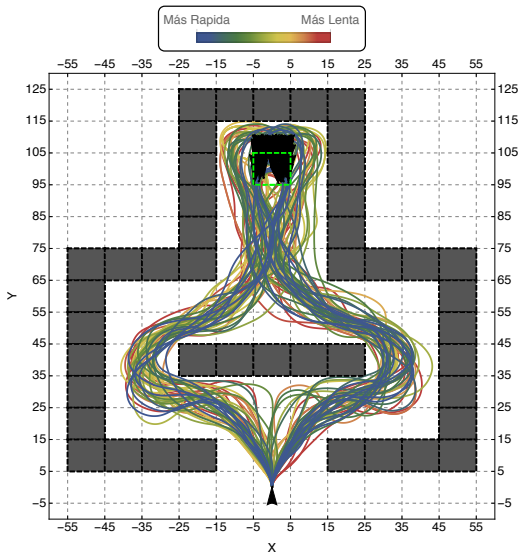


Figura 5.56: Mejores trayectorias encontradas en cada uno de los experimentos.

.

En la figura 5.57 se puede ver la mejor trayectoria encontrada. Se puede apreciar la complejidad de la maniobra en la variabilidad de las órdenes de consigna que se realizan, incluyendo el giro final para situar el barco en el objetivo con el apuntamiento correcto: 270^o.

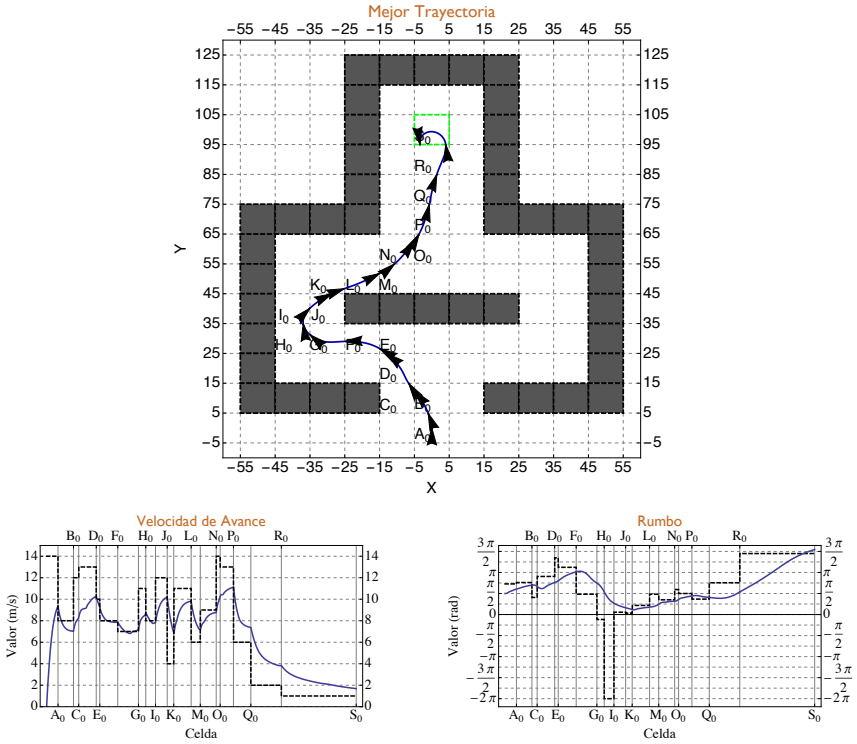


Figura 5.57: Mejor trayectoria encontrada, las sub-figuras muestran las consignas de rumbo y velocidad que describen la trayectoria: la línea azul contínua muestra la evolución del barco y la línea discontínua negra muestra los valores de consigna.

.

Escenario 4

Este escenario representa una maniobra real en una zona de costa, obtenido a partir de los datos barimétricos [MJE13] reales facilitados por la universidad de California y la NASA.

Las condiciones iniciales son:

$$[x_\tau = 0, y_\tau = 200, \theta_\tau = 180^o, v_{l\tau} = 0m/s]$$

Las condiciones finales son:

$$[x_\tau = 0, y_\tau = 100, \theta_\tau = 135^o, v_{l\tau} = --]$$

La figura 5.58 muestra el conjunto de trayectorias encontradas para realizar la maniobra. Realmente la maniobra es bastante menos complicada que en los escenarios anteriores, que son artificiales. La única complejidad es que comienza muy cercana a un obstáculo lo que restringe los movimientos de partida, pero una vez superado el escollo inicial, existe suficiente espacio como para que el barco realice la maniobra sin problemas.

En la figura 5.59 se puede ver la mejor trayectoria encontrada. Se puede apreciar cómo el giro para superar el islote final se planifica durante varias casillas

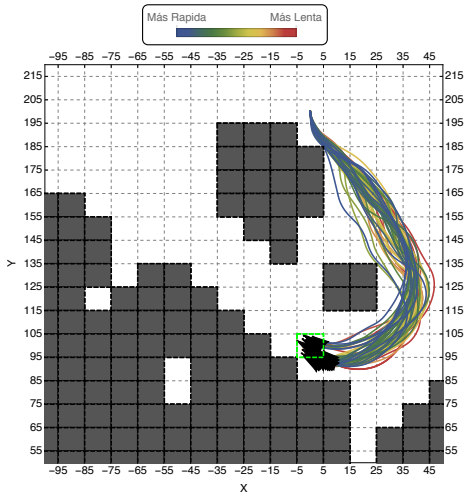


Figura 5.58: Mejores trayectorias encontradas en cada uno de los experimentos.

.

lo que permite trazarlo suavemente y colocar el barco en el objetivo sin necesidad de cambios bruscos en la trayectoria.

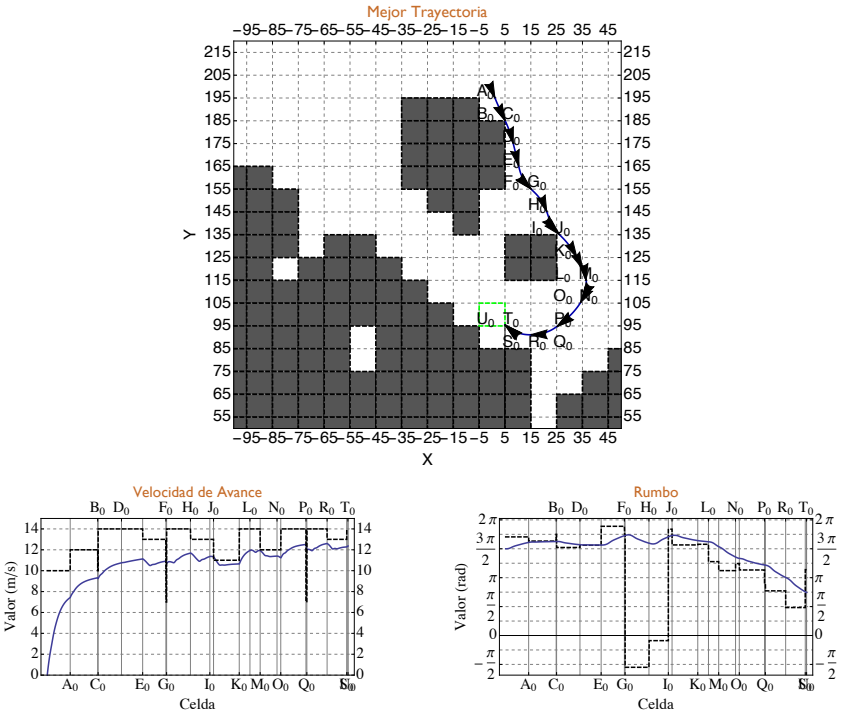


Figura 5.59: Mejor trayectoria encontrada, las sub-figuras muestran las consignas de rumbo y velocidad que describen la trayectoria: la línea azul contínua muestra la evolución del barco y la línea discontínua negra muestra los valores de consigna.

.

Discusión

Como demuestran los resultados, ACE es capaz de resolver satisfactoriamente la planificación de maniobras bajo restricciones diferenciales. En todos los casos probados, el algoritmo ha sido capaz de encontrar una forma de maniobrar viable, cumpliendo con las restricciones del escenario, y mejorarla hacia una maniobra óptima en tiempo consumido.

Lo más interesante de esta aplicación, son los desarrollos futuros. En este caso consideramos dos vías particularmente interesantes:

- Planificación en tiempo real para vehículos autónomos.

  Teóricamente es posible utilizar el método descrito para que un vehículo autónomo sea capa de planificar sus propias trayectorias, pero existen dos elementos críticos. El algoritmo requiere una representación del escenario, aunque suene obvio es necesario conocer el terreno donde uno quiere moverse antes de planificar una trayectoria.

  Un vehículo autónomo se mueve en muchos casos en terreno desconocido, lo que exigiría algún tipo de mapeo previo a aplicar el algoritmo de planificación. Ahora bien, el mapeo exige moverse, explorar, etc... por tanto es muy probable que aún no siendo óptimo, el propio mapeo lleve al vehículo a donde quiera ir sin necesidad de planificación ninguna.

  El otro elemento crítico es el tiempo de cómputo. La planificación requiere un coste computacional nada despreciable, una primera solución aceptable se puede alcanzar en minutos. Si el vehículo cambia de posición a gran velocidad puede ser que la planificación no sea viable, pues los cambios de posición invalidarían la planificación. Se podría planificar hacia futuro, extrapolando la posición al cabo de unos minutos o alguna solución similar. Pero una aplicación directa no es viable, es necesario un estudio previo.

- Planificación de flotas. El segundo escenario interesante consiste en la planificación de varios vehículos conjuntamente. Cada vehículo tiene sus condiciones de maniobra especificas y representa un obstáculo móvil frente al resto.

  Este caso sería una planificación “offline”, requiriendo menos estudio que la planificación en tiempo real. De hecho modificando la función de coste, se podría plantear una flota de exploración que maximizase el área cubierta por diferentes vehículos en un determinado lapso de tiempo.

  La complejidad de la búsqueda aumentaría, pero para el algoritmo sería lo mismo, lo único es que ahora dispone de un modelo que representa una flota y no un barco individual, las órdenes de consigna serían una lista de valores; pero volvería a ser una búsqueda en un espacio de estados.

Por último hay que señalar que se trabaja con un barco porque tiene unas restricciones dinámicas claras, pero la planificación es independiente del modelo. El mismo método se podría aplicar a un petrolero, una zodiak, o un coche, etc... Lo que posibilita también planificación mixta de vehículos.

5.4. Recapitulación

El algoritmo desarrollado, ACE, se ha probado satisfactoriamente en tres problemas completamente distintos: el viajante de comercio, un problema de optimización combinatorial clásico; programación genética, donde se persigue la obtención de expresiones formales que satisfagan un determinado predicado; y la planificación de maniobras para barcos incluyendo restricciones dinámicas. En los problemas de programación genética y el viajante de comercio, ACE se ha mostrado competitivo respecto de los demás algoritmos comparados, inclusive mejorando sus resultados.

Obviamente, disponer de una heurística adecuada es capital a la hora de resolver un problema. Como puede verse en el análisis experimental, ACE es capaz de hacer uso de heurísticas adecuadas (TSP) o aproximadas (planificación de barcos), y de este modo mejorar su rendimiento de búsqueda. Igualmente, el algoritmo es capaz de encontrar soluciones cuando la heurística sea inexistente o casi inexistente, como es el caso de la programación genética.

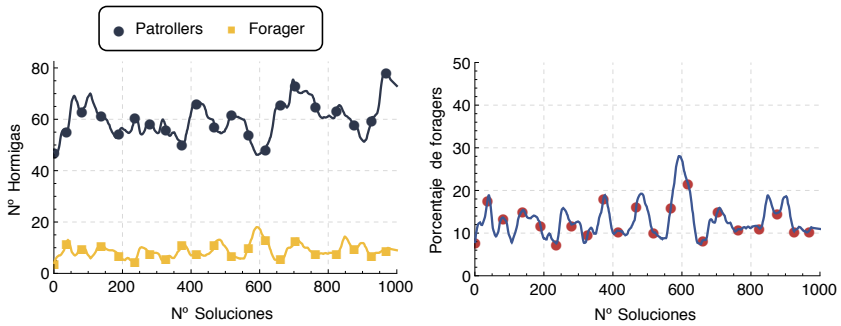
El enfoque basado en la auto-organización, permite diseñar un algoritmo muy flexible. Prácticamente, solo es necesario dar una representación adecuada del espacio de estados de cada problema, el propio algoritmo se adapta al problema. Este carácter se pone de manifiesto en la poca variación que presentan los parámetros propios del algoritmo –no vinculados a la representación– de un problema a otro:

Parámetro	TSP	P. Genética	Maniobras
γ_1	0,99	0,9	0,99
γ_2	0,5	0,5	0,5
γ_3	$400 \cdot nc$	∞	∞

La capacidad de búsqueda de ACE depende del estado de la población, la cuál se adapta según el problema. En la figura 5.60 se muestra un ejemplo de como la población varía dependiendo del problema que esté resolviendo En el TSP, construir una solución requiere 51 pasos (tantos como ciudades), lo que da lugar a un espacio de búsqueda considerable y por tanto se necesita una población con un alto número de agentes. En el caso de la programación genética, el tamaño de las soluciones varía considerablemente, por este motivo, la población presenta un comportamiento más complejo que en el caso del TSP. En el caso de la planificación de maniobras, ocurre igual que en la programación genética: el tamaño de las soluciones oscila. A su vez encontrar maniobras viables puede resultar difícil, lo que da lugar a unas oscilaciones más pronunciadas en la población de patrollers.

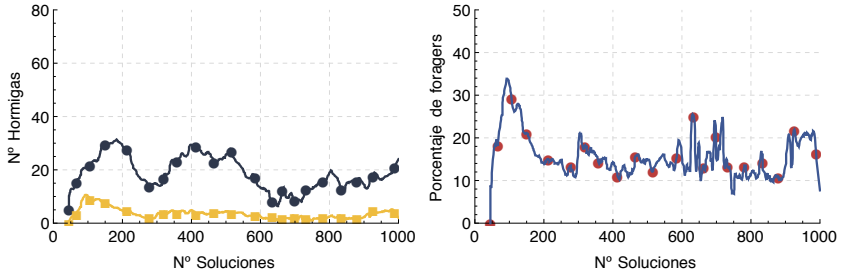
En todos los casos de la figura 5.60, se puede apreciar como el balance entre las poblaciones oscila. Según sea necesario adquirir nueva información, se incrementa el número de patrollers para forzar la exploración de nuevas soluciones. En cambio, cuando se desea explotar la información adquirida, se reduce el número de patrollers y se aumenta el número de foragers.

El resultado es un algoritmo flexible y robusto que es capaz de resolver un problema auto-organizándose en función del estado de la búsqueda. Combinado con el hecho de que una representación en espacio de estados es a su vez una forma "universal" de representar problemas, obtenemos un algoritmo de propósito general que puede ser aplicado de manera sencilla a diferentes problemas.



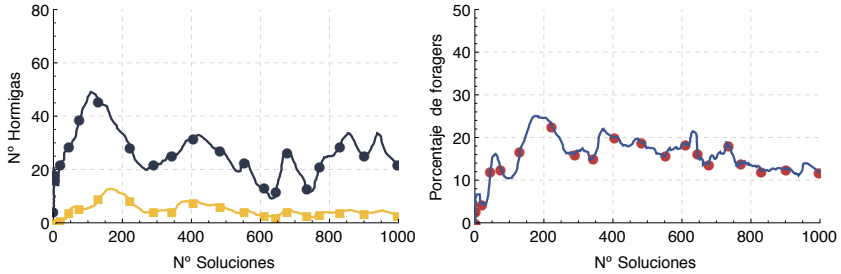
(a) Instancia del TSP de 51 ciudades.

(b) TSP, porcentaje de foragers.



(c) Programación genética: "Santa Fe, artificial ant trail".

(d) Programación genética, porcentaje de foragers.



(e) Planificación de una maniobra de 100m con obstáculos.

(f) Planificación de una maniobra, porcentaje de foragers.

Figura 5.60: Ejemplo de la evolución de la población durante las 1000 primeras soluciones para los distintos problemas evaluados.

Capítulo 6

Conclusiones

A lo largo de esta investigación se ha mostrado detalladamente cómo diseñar un algoritmo meta-heurístico utilizando una aproximación de computación emergente. El diseño se inspira en las capacidades de auto-organización de las colonias de insectos. Esta auto-organización consiste en ajustar la conducta en función de la respuesta obtenida al realizar una acción en un determinado entorno. Este mismo esquema lo trasladamos a un algoritmo de búsqueda: las acciones serían la generación de soluciones y el entorno sería el problema que se desea resolver. De este modo se consigue que el algoritmo se auto-organice según el estado de la búsqueda.

- Para desarrollar el algoritmo hemos partido de la meta-heurística ACO: un sistema multi-agente, donde los agentes se coordinan mediante la información compartida en una estructura de datos común, la tabla de feromona. Este esquema se modifica para aplicar un enfoque clásico de la Inteligencia Artificial: búsquedas en espacios de estados. El resultado es un sistema capaz de explorar el espacio de estados de un problema aprendiendo a construirlo. Es decir, un sistema capaz de explorar nuevas soluciones a un problema, a la vez que optimiza las ya encontradas.

- Inspirándonos en los procesos de toma de decisiones colectivas de las colonias de insectos, obligamos a los agentes del sistema a operar de manera asíncrona. A su vez, modificamos la función de la tabla de feromona, para que actúe más como un dispositivo de comunicación que como un almacén de información. Al ser la tabla un canal compartido, el flujo de información es limitado y la conectividad entre los agentes se reduce.

  La baja conectividad hace posible distribuir la información entre la población de agentes. De este modo, reducimos la influencia a la que se ve sometido cada agente, incrementando su libertad de converger a un patrón distinto que el mayoritario. Lo cual se traduce en un mejor proceso de búsqueda, al reducirse el riesgo de estancarse por una pérdida de diversidad.

- Para conseguir que el sistema busque de manera eficiente es necesario alcanzar un equilibrio en la búsqueda, que ésta no sea excesivamente dispersa, ni demasiado restringida. Es decir, de algún modo hay que controlar el sistema. De nuevo nos inspiramos en la naturaleza: partiendo de las dinámicas auto-organizativas típicas de las colonias de insectos, hemos desarrollado

una solución para permitir que el algoritmo sea capaz de controlarse a sí mismo.

Introducimos en la población dos tipos de agentes, un primer tipo (patrollers) aumenta la diversidad y un segundo tipo (foragers) la reduce. Desarrollamos una dinámica auto-organizativa para regular la población de agentes. Esta dinámica de población consiste en un reclutamiento: cada agente, en función de su rendimiento, recluta o inhibe un tipo de agente. Si un agente tiene éxito en la tarea asignada, recluta un agente del tipo contrario para mantener un equilibrio. En caso de no tener éxito, recluta un agente de su mismo tipo.

La dinámica también permite al sistema auto-regular el tamaño de la población. Cuando es necesario incrementar la diversidad en la búsqueda, se favorecen poblaciones grandes. Cuando conviene reducir la diversidad, se reduce el número de individuos.

- Estas técnicas de diseño, disminuyen el uso de parámetros de un algoritmo, a la par que reducen la influencia de los parámetros en el rendimiento del mismo. El valor de un parámetro deja de ser algo crítico, puesto que el algoritmo se auto-organiza. También es importante advertir que el uso de la auto-organización comparada con el uso de parámetros, puede incrementar el rendimiento de un algoritmo al permitir variar el comportamiento del algoritmo a lo largo de la búsqueda.

Todo este conjunto de ideas se materializan en la implementación de un algoritmo que hemos llamado ACE, el cual se ha sometido a diferentes análisis experimentales:

a) Resolver el problema del viajante de comercio, comparando el rendimiento de ACE con dos algoritmos ACO clásicos: ACS y MMAS.

   El resultado de este experimento muestran que, en general, ACE ofrece un rendimiento significativamente superior a ambos algoritmos.

b) Resolver problemas clásicos de programación genética, comparando su rendimiento con un algoritmo genético simple.

   Los resultados de este experimento muestran que el algoritmo es capaz de resolver los problemas estudiados. En comparación con un algoritmo genético, dependiendo del problema estudiado, ACE ofrece un rendimiento superior o inferior. En programación genética, la comparativa se hace atendiendo al esfuerzo computacional: numero de expresiones procesadas para satisfacer un predicado. Los resultados muestran que las expresiones de ACE son significativamente más pequeñas que las del algoritmo genético. Por tanto habría que considerar el tamaño de las expresiones a la hora de calcular el esfuerzo computacional.

c) También hemos comparado ACE con técnicas del estado del arte actual para la aproximación de expresiones matemáticas. En estos casos, se utiliza la evaluación de nodos y no expresiones completas para contabilizar el tiempo de búsqueda. Los resultados muestran que ACE es claramente superior a los demás algoritmos comparados.

d) Optimización de maniobras para barcos autónomos.

Este último ensayo consiste en la aplicación de ACE a un problema real. El problema implica realizar integraciones numéricas del modelo diferencial del barco para satisfacer sus restricciones dinámicas, lo que conlleva un coste computacional alto. Esto obliga a resolver el problema evaluando muy pocas soluciones en comparación con otros problemas típicos de optimización. Los resultados muestran que ACE es capaz de obtener maniobras cercanas a la solución óptima con un número de integraciones reducido.

Como se ha demostrado en esta investigación, la computación emergente es una alternativa viable y eficaz para el diseño de algoritmos, permitiendo el desarrollo de entornos flexibles y robustos, sin necesidad de procedimientos complejos.

6.1. Trabajos futuros

En el estudio realizado se pueden distinguir dos vías principales de investigaciones futuras:

i) Continuar la aplicación de ACE a otro tipo de problemas.

   Los enfoques de auto-organización muestran su potencial frente a otras técnicas, cuanto más libertad de búsqueda ofrezca el problema. Un ejemplo de este tipo consiste en problemas de diseño automático, donde cada tipo de problema es distinto a otro y por tanto hacen difícil el desarrollo de técnicas específicas.

   Otro marco interesante lo constituyen los problemas de optimización continuos. Un primer estudio aproximado [EJG10a] muestra que sería posible una aplicación de ACE a este tipo de problemas.

ii) Aplicar los esquemas de auto-organización desarrollados en otro tipo de algoritmo.

   ACE es el resultado de implementar una serie de ideas, las cuales se pueden extrapolar a otros contextos u algoritmos, como pudiese ser el Particle Swarm Optimization (PSO). Podría ser viable aplicar un manejo de las poblaciones, similar al que se lleva a cabo en ACE, para permitir a las partículas auto-organizarse. De este modo, se podría minimizar el uso de parámetros, los cuales en el caso del PSO son difíciles de ajustar.

iii) Explorar otros métodos de auto-organización.

   La inclusión de una dinámica de poblaciones abre una vía de estudio que consiste en explorar nuevas formas de relacionar las poblaciones de cara a incrementar la eficacia del algoritmo.

Capítulo 7

English summary

Introduction

The proposal of this thesis is to design a search algorithm capable of solving problems in a intelligent way, where this intelligence arises from the dynamics of the algorithm, i.e., it is a form of emergent computation.

Quoting S. Forrester [For90], we can define the term "emergent computation" as follows:

(i) A collection of agents, each following explicit instructions

(ii) Interactions among the agents (according to their instructions), which form implicit global patterns at the macroscopic level, i.e. epiphenomena

(iii) A natural interpretation of the epiphenomena as computations

A typical example of emergent computation is the cellular automata (CA) [Lan90, Wol02]

Formally, a cellular automata is defined as a matrix of dimension $n \times m$ where each cell of the matrix is an automaton. Each automaton receives as inputs the states of adjacent automata, where this region is known as the neighbourhood of the automaton. Thus, the state transition of each automaton is a function of its neighbours.

For instance, a cellular automata of two dimensions, where the neighbourhood consists of 8 automaton, can be represented as follows:

a_0	a_1	a_2
a_3	x	a_4
a_5	a_6	a_7

the transition function (δ) which determines the state of x, is given by the states of its neighbours: $\delta(x) = f(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7)$. Depending on the definition of δ, it is possible to obtain different kind of cellular automata.

For example, a well-known cellular automata is the "majority rule". Each automaton can have two states: 1 (favourable vote), or 0 (unfavourable vote). Firstly, each automaton is initialised by random. After that, it is proceed with evolution as follows:

- $state = 0$ if in the neighbourhood of the automaton the overwhelming preference is 0 : (No. 0 >No. 1)
- $state = 1$ if in the neighbourhood of the automaton the overwhelming preference 1 : (No. 1 >No. 0)
- In case of a tie (No. 0 = No 1) the state remains unchanged.

In figure 1.1, it is represented an example of the "majority rule" automata for a neighbourhood of 8 automaton. As the automata in the same neighbourhood tend to converge to similar states, it can be observed the emergence of clear patterns of opinion.

Cellular automata are a form of emergent computation. Cellular automata show a tendency towards the formation of patterns: large stable groups of automata where all are in the same state. These patterns are an emerging phenomenon because they are not programmed in the set of instructions that define an automaton. The order exhibited by the cellular automata as a whole, is the product of the relationship between the automaton.

The emergent computation approach can be used to tackle the critical aspect of the design of metaheuristic algorithms[BR03]:

> In short we could say that metaheuristics are high level strategies for exploring search spaces by using different methods. Of great importance thereby is that a dynamic balance is given between diversification and intensification. The term diversification generally refers to the exploration of the search space, whereas the term intensification refers to the exploitation of the accumulated search experience.

This balance of the search can be explicitly implement, but it can also be implemented through emerging computing. The control mechanism would not be explicit, but an emerging system capacity. If the algorithm is designed as a multi-agent system, then it is possible to achieve a search balance through the interactions among the agents.

7.1. Preliminary framework

The first stage of design can be summarised as the combination of the Ant Colony Metaheuristic (ACO)[DS04] with a representation of the problems using a state space.

ACO meta-heuristic is already a multi-agent system. Thus it is a reasonable choice in order to design a system where the dynamics comes from the interactions among the agents. Besides, a multi-agent system is a typical technique to study self-organised behaviour in nature.

Regarding the use of a classical representation in a state space, the reasons are two:

i) A graph can be a complicated representation depending on the kind of problems tackled.

   A graph representation requires to be built prior to the application of the algorithm, if the graph has a remarkable size, then there could be performance issues regarding the computation time and the algorithm convergence. The

initial pheromone is a flat distribution of probability. The algorithm makes this distribución to converge towards a few choices.. If such distribution has thousands of options (each node has many neighbours), convergence can be quite complicated.

Using a state space methodology, it is possible to tackle other class of problems. In problems where the graph is not an adequate representation, it is possible that a state space can be. Since a graph is a way to define (to represent) a state space, this change does not mean a loss of generalisability.

Besides, the state space does not need to be previously constructed, since the search itself consist of building this space. This implies that the search no longer starts from a flat distribution, but from an "empty" distribution that is filled along the search. Therefore, it can be more easy to avoid performance issues in complex problems (computation time), and at the same time it decreases the convergence issues.

ii) The state space approach allows us to define public information as two separate distributions: one static (heuristic information), and one dynamic (pheromone information).

In ACO, these are not distributions, but component to define a way to take a decisions. In our case, these distribution are independent in terms of taking a decisions: any of them can be used in a isolated way as information source to take a decision, but they can not be used together.

This separation allows expressing the problem of finding a search balance (exploration / exploitation) in terms of activity: depending on the frequency of use of each type of information, heuristics or pheromone, the algorithm search can be more or less dispersed.

As the framework is a multi-agent system, we can define two kind of agents depending on which information source they use to take decisions. In this way, the search balance can be tackled as a problem of how to distribute the agents of the system.

An scheme of this preliminary framework is shown in figure 7.1, where it can be observed how the relationship among the agents is through the problem and the public information (pheromone).

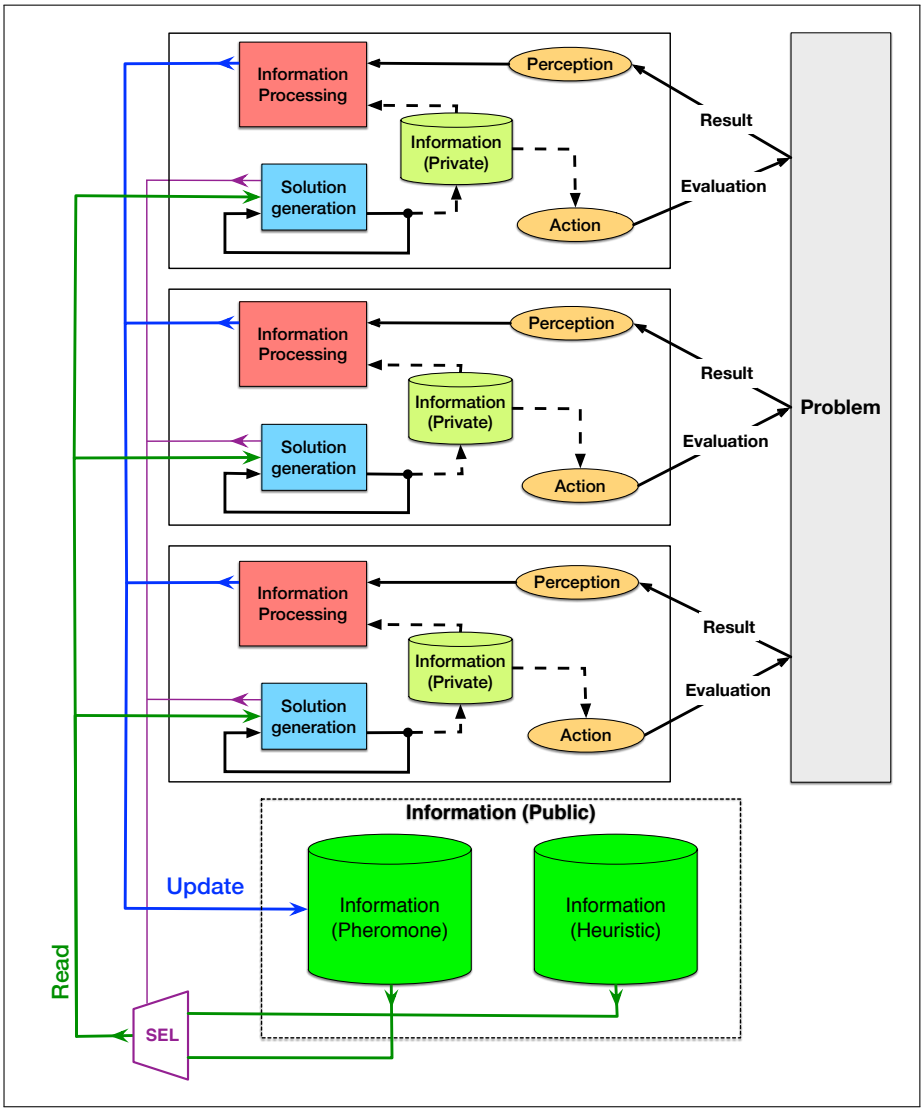


Figura 7.1: Basic outline design multi-agent system.

7.2. Collective decisions in social insects

There are two interesting models of collective decision making in social insects: ant colonies and bee colonies. The purpose is to combine elements from both models in order to strength the algorithm design. In order to fulfil this purpose, in the following sections it will be highlighted the fundamental aspects of each model.

Ants: local information

The ants decisions are based on pheromone mark, which is deposit in the ground while the ants are moving around. The most interesting thing of this scheme is that this way the information is associated to each singular decision. This means that in order to carry out a certain behaviour, an ant does not have information about the global pattern of conduct. Instead, the information is fragmented: each time it has to take a decision, the ant has information available regarding to the decision to be taken.

This form of association of information to each individual decision is that we refer as " local information ".

The main advantage is that it allows to combine different sources of information obtained by different individuals. Since the information only refers to a single action, not a complete behaviour, it is possible to develop a new behaviour just combining decisions that comes from different individuals.

Bees: temporal information

The most interesting characteristic of bees is their ability to change the source of nectar that they are exploiting if a better one is discovered. Like ants foraging pattern converge to a certain path, the decision process of bees converge to a certain solution. However, if the conditions change, the bees are able to alter their decision, instead, the ants are unable to change it.

The difference between these two behaviours is mainly explained by the process of communication used by each kind of insect colony:

- In ants communication is through the environment: an ant leaves a mark in accordance with the decision. This causes that the information is attached, so it does not disappear.

  This produces a rapid convergence towards a certain pattern, for example a line of ants. However, this also means that the pattern is stable. The accumulation rate of pheromone in the predominant pattern is so high that prevents any significant accumulation of it in any other behaviour pattern.

  For example, when an ant tends to get out of the row, the influence of other ants (which remain in the row) masks any pheromone trace left by this ant, simply because there are many ants in the row.

- In bee colonies, all information is acquired in a single communication event. After acquiring it, the bee is ready to use it in order to exploit a nectar source.

  In bee colonies, the behaviour is not acquired as local information attached to each decision. Bees acquire all information in a single communication

event. After acquiring it, the bee is ready to use it in order to exploit a nectar source. The information is shared by one bee to another. This means that the communication has a period, once it ends the information can not be acquired by anyone else. Thus it is possible for a colony to exploit multiple nectar sources at a time.

The bees works asynchronously. This makes possible that one bee communicates information about one nectar source at time t, while other bee communicates information regarding to other source at time t'. Since the communication only holds for a certain time, it is possible that both communication processes never overlap each other. Since the communication does not overlap, the recruitments remain independent one from each other, and the colony can exploit both sources.

This makes a very flexible collective decision dynamics because the majority pattern does not need to influence the minor ones, since everything occurs at different times. Thus the colony can maintain different options at a time, focusing on the better one, but not completely discarding the minority options. In such way, if the situation changes, the bee colony is able to alter the global pattern of foraging.

Each biological model has its advantages and disadvantages. Ants are able to combine information between them, but there is no temporal component that provides the flexibility to change a decision already made. Conversely bees can easily change a decision already made, and even settling on several simultaneous choices. In contrast, their information about a behaviour is global.

The idea is to combine both models developing a hybrid scheme. The key is to convert the pheromone table of ACO on a " buffer ": a data structure reserved for temporary storage, waiting to be processed. In this way it is combined the local information of ants with the temporal dynamics of bees.

Experimental example

A simple example of this hybrid scheme can be shown through simulations. The code of the algorithm is shown in figure 7.2. The example consists in a simple multi-agent system where each agent has to generate a sequence of symbols.

The system has a table which holds the information in pairs of sequence position-symbol: for instance, $4 \rightarrow A$. This means that at the position 4, an agent should write the symbol A.

Once an agent finishes the construction of a sequence, it updates the table by overwriting the table contents. This means that firstly, it wipes out the table content. Afterwards, it fills the table according to the sequence it has generated.

There are four agents (A,B,C,D), and the length of a sequence is 4 symbols. We can initialise them synchronously (typical scheme of ACO), but also they can be initialised asynchronously. The initialisation is shown in table 7.1.

```
1: agent = Initialisation
2: table = []
3: while i <N do
4:     for a ∈ agents do
5:         if end(a) then
6:             table = update(a)
7:             a = []
8:         end if
9:     end for
10:    for a ∈ agents do
11:        a = symbol(table,a)
12:    end for
13:    i = i + 1
14: end while
```

Figura 7.2: Simple algorithmic scheme.

	Asynchronous				Synchronous			
Agent A	A_0	A_1	A_2	A_3	A_1	A_1	A_2	A_3
Agent B	B_0	B_1	B_2	-	B_0	B_1	B_2	B_3
Agent C	C_0	C_1	-	-	C_0	C_1	C_2	C_3
Agent D	D_0	-	-	-	D_0	D_1	D_2	D_3

Tabla 7.1: Agents initialisation.

The results of the simulation are shown in figures 7.3 and 7.4. It can be seen that asynchronous scheme can develop a large collection of sequences. This is possible because the agents does not converge to the same sequence.

This scheme provides a more flexible system that reduces the risk of losing search diversity, making a better searching process. However, it needs some form of control in order to achieve a balance of the search. In order to do this, we will "imitate" the self-organising processes of an ant colony.

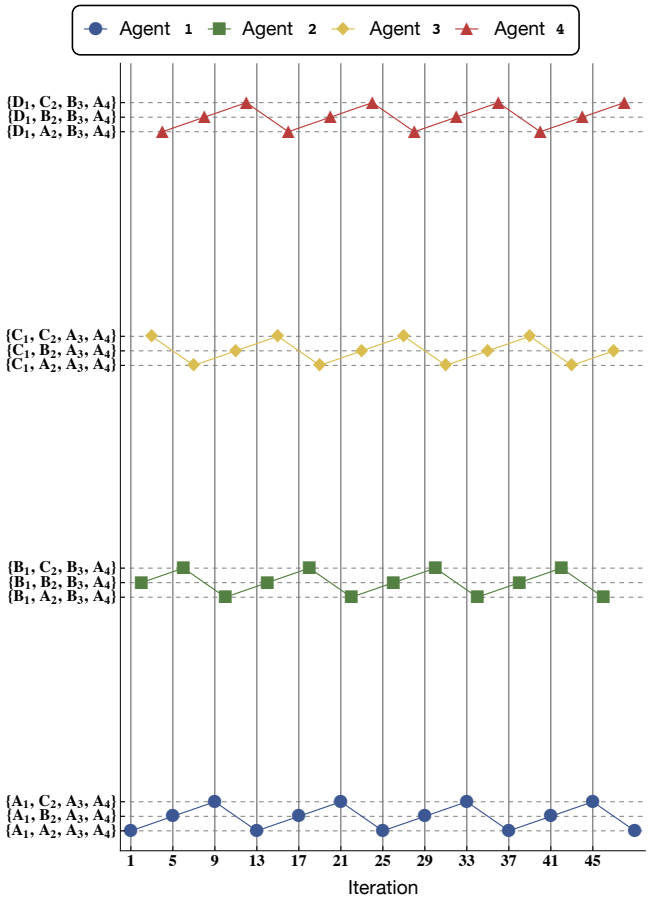


Figura 7.3: Evolution of agents for asynchronous model.

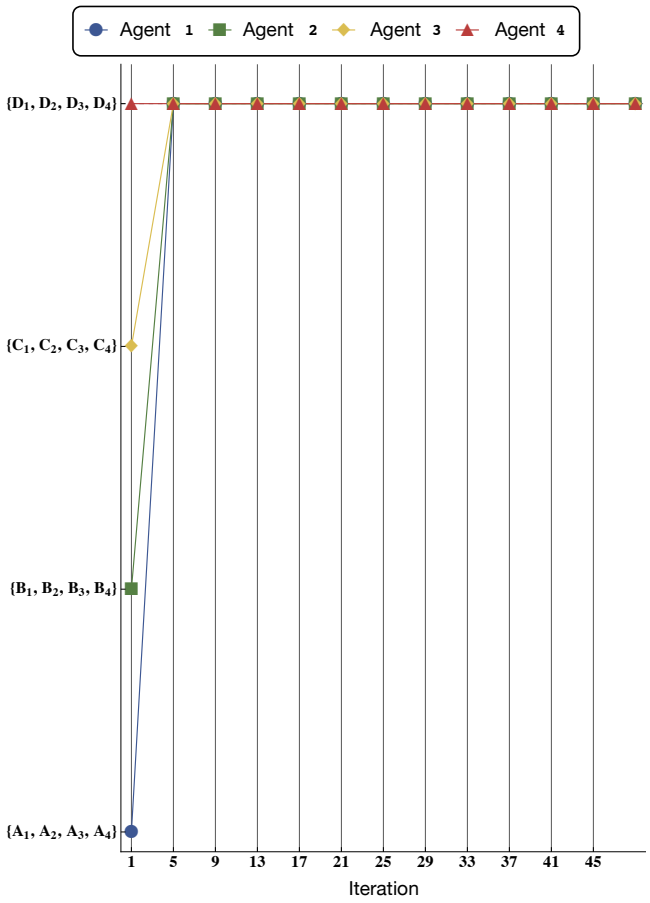


Figura 7.4: Evolution of agents for synchronous model.

7.3. Self-organising dynamics

Biological researcher Deborah Gordon [Gor10, Gor99] reveals that a colony self-regulates its organisation by the interactions or *encounters* among the ants, i.e. direct communication. The organisation of a colony refers to how the ants are distributed among the different tasks that must be undertaken. It is based on roles –behaviour patterns–, where each role is associated to the performance of a specific task[1].

Especially interesting is her description on how the foraging activity is regulated. Gordon points out that there are two kinds of ants:

- Patroller ants: their task is to locate the food sources before the foraging activity begins.
- Forager ants: their task is to exploit the food sources discovered.

How an ant is assigned to one of these tasks, arises from encounters. For example, once patroller ants locate the food source, they return to the colony and interact with those ants that are at the entrance of the colony. These last can react to encounters with patroller ants and start gathering food, i.e. they become foragers. Thus, the total number of foragers is determined by the rate of encounters with patroller ants.

The pool of unassigned workers, i.e. the nest, allows to distribute the available ants among the tasks to be performed: each unassigned ant can perceive an stimulus associated to one task and reacts to it, i.e. the ant starts to perform the task. Thus the nest acts as the "meeting point" where all the stimuli converge. The stronger the stimulus, the greater the number of ants associated to it. But an ant does not perform a task for a long period of time. Instead, it tends to come back to the nest, becoming inactive as soon as possible. Only if the stimulus is adequate the ant is re-engaged to its previous task. This prevents the ants to get engaged to non-priority tasks, allowing fast changes in the organisation.

For example, an inactive ant starts to forage only if it interacts with another ant. So each forager that comes back to the colony with food, does not start to forage again immediately; it has to be stimulated by a patroller ant which has found food or by another returning forager carrying food. In this example, the stimuli are related to the rate of success in task completion.

The foraging activity of the colony works by a positive feedback: a higher success in finding food increases the number of ants assigned to foraging. As more ants are assigned, the chance of success also increases. Thus a stable pattern of activity emerges and the colony self-organises. But since the activity requires a "continuous" reinforcement of interactions, as soon as the food source begins to be exhausted, the reinforcement decays and the pattern of activity starts to disappear, allowing the forager ants to be engaged to another tasks.

In her studies, Gordon points out that the forager ants only start foraging if their rate of encounter with patroller ants is higher than a threshold, sporadic encounters do not seem to stimulate forager ants. A high rate of encounters means that there is abundant food available, so, the foraging activity is worthwhile. A low rate means that the food is scarce or too sparse, so, the foraging activity is not worthwhile. This description can be translated to an algorithm context: a high

[1] Although, all ants are identical, it is common to differentiate kinds of ants according to different roles.

rate of success of exploration search means that it is worthwhile some exploitation search in order to reduce the dispersion, a low rate of exploration success means that the exploitation search is not worthwhile.

Finally, the spontaneous activation of ants is needed to guarantee activity; since the stimulus is the activity, if there is no activity, there is no recruitment. For this reason some ants can become active spontaneously due to environmental causes. For example, patroller ants activity does not depend on recruitment, the patroller ants become active every day in the morning.

In the context of an algorithm, it is also possible to consider these two tasks. Exploration can be assigned to one kind of agents: patrollers. Exploitation can be assigned to other kind of agents: foragers.

Population Dynamics procedure

This procedure implements a mechanism for regulating exploration and exploitation.

In a first stage, once an ant has completed the construction of a solution it is removed from the population of active ants. Depending on whether the ant has succeeded or not, it activates the procedure Success(ant) or the procedure Failure(ant). These procedures determine the kind and number of ants to be recruited.

In a second stage, once the whole population of active ants has been processed, the prescribed number of ants are recruited using the procedure Recruitment().

These three procedures share the following common set of variables (counters),

- RP, number of patrollers to be recruited.
- RF, number of foragers to be recruited.
- UP, number of unsuccessful patrollers since the last successful one.
- SP, number of successful patrollers since the last successful foragers.
- FR, current number of active foragers.
- PR, current number of active patrollers.

Success

Figure 7.5 shows the Success(ant) pseudocode.

A successful forager recruits a patroller and resets to zero the counter SP.

A successful patroller increases the counter SP, resets the counter UP and recruits a patroller or a forager depending on the number of foragers already active and the number of successful patrollers counted by SP.

Failure

Figure 7.6 shows the Failure(ant) pseudocode.

An unsuccessful forager limits itself to decrease the counter of active foragers. On the other hand, an unsuccessful patroller first increments in one the counter of unsuccessful patroller UP and then, recruits patrollers according to the value of UP, using the function: $round(log(UP + 1))$.

```
1: procedure Success(ant)
2:     if ant is Forager then
3:         FR ← FR − 1
4:         SP ← 0
5:         RP ← 1
6:     else
7:         PR ← PR − 1
8:         SP ← SP + 1
9:         UP ← 0
10:        if SP ≥ FR then
11:            RF ← 1
12:        else
13:            RP ← 1
14:        end if
15:    end if
16: end procedure
```

Figura 7.5: Pseudo-code description: Success case.

Lastly, the procedure, no matter which kind of ant it has been applied to, calculates and normalises the difference between μ and $g(best)$. If this quantity is greater than the rate of foragers in the active ant population, a new forager is recruited.

```
1: procedure Failure(ant)
2:     if ant is Forager then
3:         FR ← FR − 1
4:     else
5:         PR ← PR − 1
6:         UP ← UP + 1
7:         RP ← round(log(UP + 1))
8:     end if
9:     if (μ − g(best))/μ ≥ FR/(FR+PR) then
10:        RF ← 1
11:    end if
12: end procedure
```

Figura 7.6: Pseudo-code description: Failure case.

Recruitment

Figure 7.7 represents the procedure of recruitment. Until a first solution has been found, a patroller is released at every cycle. This is particularly useful for those problems in which finding feasible solutions is part of the problem. Once

this initialisation phase ends, the population is updated according to the values of RP and RF.

```
1:  procedure RECRUITMENT
2:      if samples < 1 then                          ▷ Initialisation phase.
3:          Population ← createPatroller()
4:          PR ← PR + 1
5:      else
6:          while RP > 0 do
7:              Population ← createPatroller()
8:              PR ← PR + 1
9:              RP ← RP − 1
10:         end while
11:         while RF > 0 do
12:             Population ← createForager()
13:             FR ← FR + 1
14:             RF ← RF − 1
15:         end while
16:     end if
17: end procedure
```

Figura 7.7: Pseudo-code description: Recruitment and initialisation.

Self-regulating exploration and exploitation

The pheromone table contains a probability distribution of actions to perform at each state. The foragers tend to make this distribution converge towards a single action. In contrast, the patrollers tend to spread the probability over different actions. For instance, a patroller may add new actions that previously were not present in the pheromone table.

Usually, the solutions provided by foragers are not significant: they just repeat solutions that were first discovered by patrollers. They are suitable instead for redirecting and maintaining the search into promising areas of the solution space. Patrollers, meanwhile, are suitable to discover new solutions, but they can easily disperse the search far from these promising areas. The population dynamics attempts to coordinate patroller and forager populations in order to maintain the search capacities of the patrollers bounded into promising zones.

The basic way of coordination relies on the recruiting made by successful ants : they always recruit an ant from the other kind. For instance, if patrollers are succeeding, they tend to become inactive, and the number of foragers tends to increase. Once the foragers have succeeded, they recruit new patrollers. This helps to avoid patrollers disperse themselves too much, because the foragers help to converge the search towards suitable zones.

In general, a low number of foragers is enough to do the work. In addition, a high number of them can lead the search to stagnation. That is why the mechanism to recruit new foragers is ruled by the number of them already active. Besides,

their number is also adjusted using the mean and the best-so-far solution, which is a rough estimation of the algorithm convergence.

Patrollers, in contrast, are recruited whenever a forager succeeds and, also, when they are not able to find good enough solutions. In this last case, the exploration is failing and it is mandatory to reinforce it.

The behaviour of the population dynamics varies from problem to problem. An interesting example can be found in [EJGS13] where, after finding a suboptimal solution, foragers and patrollers cooperate to improve it as much as possible. Later on, the algorithm jumps to another solution, found by patrollers, avoiding stagnation.

7.4. Experimental summary

Departing from the ideas of the previous sections, it has been developed a new ant algorithm: Ant Colony Extended (ACE)[2]

In order to evaluate ACE, it is applied to solve various problems. Since the algorithm is based on self-organisation, we have chosen three radically different problems instead of considering similar problems. This principle allows to achieve good performance in different search environments without particularising the algorithm for each type of problem. Of course, it needs the definition of an adequate representation.

The selected problems are:

i) Comparison with ACO algorithms using the TSP as a benchmark. The aim of this study is to evaluate the performance of ACE on a typical benchmark of ant algorithms.

   The results of this study can be consulted in 5.1 and in [EJGS15].

ii) Application to genetic programming problems, comparing its performance with standard genetic algorithm. The aim of this study is to evaluate ACE on a scenario radically different from TSP: developing formal expressions.

   Compared to the TSP, the most relevant characteristics of this benchmark are two: a) there is no heuristics to guide the search; and b) the solutions do not have a default dimension, instead it is is variable and unknown.

   The results of this study can be consulted in 5.2.

iii) Application to a real problem: ship planning manoeuvres. The goal is to find a sequence of discrete set-points (speed and direction) to enable the ship to a particular manoeuvre. The trajectory calculation is achieved by solving a differential model including the dynamic constraints of the vessel.

   Due to the computational cost of the numerical solution of a differential model, in order to obtain a viable manoeuvre and minimise the time required for their implementation, ACE should be able to search efficiently, since the model is only evaluated a limited number of times.

   The results of this study can be consulted in 5.3 and in [EJGS12].

[2]The full implementation can be consulted in the appendix D.

In all problems ACE has shown a remarkable performance. Besides, in the TSP and genetic programming benchmarks it has shown to be competitive with the state of arts techniques, even improving their results.

Since ACE has been designed using emergent computation and self-organisation properties, it does not require extensive configuration of parameters values. In fact, although the problems are very different among them, parameters values of ACE are very similar among them (see table below). This all os possible due to the self-organisation ability of the algorithm, which is able to adapt its population to the state of the search.

Parameter	TSP	Genetic P.	Ship planning
γ_1	0,99	0,9	0,99
γ_2	0,5	0,5	0,5
γ_3	$400 \cdot nc$	∞	∞

7.5. Conclusions

This research demonstrates (through the implementation of ACE) that emergent computation and self-organisation is a viable and effective alternative for the design of searching algorithms. It provides a flexible and a robust framework development environment without requiring complex procedures.

As future works, it is worth to mention the following research lines:

i) To study the application of ACE to another kind of problems.

ii) To study the self-organisation schemes used to develop ACE to other algorithm environments, for instance PSO,...

iii) To study other forms of self-organisation and their implementation in an algorithm context.

Apéndices

Apéndice A

Información y Entropía

La teoría de la información, es una teoría matemática propuesta por Claude E. Shannon a finales de la década de los años 1940. Inicialmente, la propuesta se llamó "A mathematical theory of communication" [Sha48], donde el foco estaba puesto en el análisis matemático para la transmisión y el procesamiento de la información. Posteriormente, la propuesta de Shannon dio origen a una rama de las ciencias de la computación que estudia la información y lo relacionado con ella.

El uso del término "información" lo asociamos a conocimiento, sentido,etc... Pero en la teoría de la información el significado del término es radicalmente opuesto.

> Frequently the messages have meaning; that is they refer to or are correlated according to some system with certain physical or conceptual entities. These semantic aspects of communication are irrelevant to the engineering problem. The significant aspect is that the actual message is one selected from a set of possible messages. [..] If the number of messages in the set is finite then this number or any monotonic function of this number can be regarded as a measure of the information produced when one message is chosen from the set, all choices being equally likely.

Como señala Shannon en su articulo, *el término información es una medida cuantitativa que expresa la variabilidad de la aparición de ciertos símbolos*. Por ejemplo, supongamos que queremos establecer una comunicación vía morse, por lo tanto tenemos que enviar letra a letra. Para enviar la palabra "queso" deberíamos enviar primero la "q", luego la "u", etc.. Pero en español después de la "q" siempre va una letra "u", por tanto no existe variabilidad y no es necesario enviar la "u". Si enviamos "qeso", ya sabemos que falta la "u". La letra "u" no aporta ninguna información, lo que diferencia la palabra "queso" del resto de las palabras en español son los demás símbolos, los que presentan variabilidad.

La información está asociada a la variabilidad, ésta variabilidad se cuantifica mediante la entropía de la información (H). *La entropía es una medida de la cantidad de información* que se calcula de la siguiente manera:

$$H = -\sum p * \log(p)$$

donde p es la probabilidad de aparición de un determinado símbolo. La entropía total será el sumatorio de la variabilidad asociada a cada símbolo.

Por ejemplo, si tenemos una variable x que puede únicamente representar dos símbolos: a y b, donde la probabilidad de cada símbolo es: $P(x = a) = p$ y $P(x = b) = 1 - P(x = a)$. La entropía asociada ($H(x)$) a la variable x en función del valor de $P(x = a)$ varía según se muestra en la figura A.1. La entropía mide la cantidad de información asociada a la variable x, y esta cantidad será máxima cuando los dos sucesos $x = a$ o $x = b$ son equiprobables porque la variabilidad es máxima.

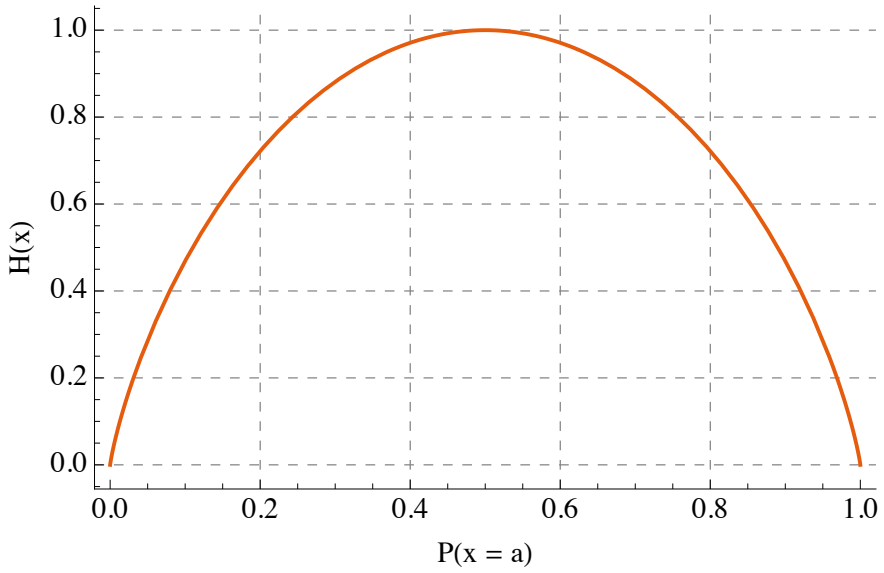


Figura A.1: Entropía de una variable x que puede tomar dos valores $x = a$ o $x = b$, en función de la probabilidad de $P(x = a)$.

La idea intuitiva detrás del concepto de información es muy sencilla. Si no sabemos que puede ocurrir, el hecho de que ocurra algo nos aporta una cantidad de información importante. Si por el contrario, un suceso es mucho más probable que otro, la información que recibimos es menor porque ya sabíamos lo que iba a ocurrir. Es decir, la información no es que $x = a$ o $x = b$, el símbolo no tiene valor. La información está asociada a la probabilidad de tomar como valor uno u otro símbolo, independientemente de cuales sean.

Cuando utilizamos un algoritmo de búsqueda para resolver un problema, el análisis se centra en la dispersión de la búsqueda. Una búsqueda excesivamente dispersa es aquella que "mira" en cualquier "sitio", por tanto las probabilidades de dar con la solución son bajas. Una búsqueda con poca dispersión (atascada) es aquella que sólo "mira" en un "sitio", igualmente si la solución no está ahí, pues no se va a encontrar. La dispersión en un algoritmo de búsqueda, no es más que la variabilidad de producir unos símbolos u otros, y por tanto se puede cuantificar utilizando la entropía de la información.

Apéndice B

Esfuerzo computacional

El *esfuerzo computacional* es una medida de rendimiento definida por Koza[Koz92] en programación genética.

> A method for measuring the performance of the genetic programming paradigm in terms of the amount of computer processing necessary to solve a particular problem. Specifically, we measure the number of individuals that must be processed in order to satisfy the success predicate of the problem with a certain specified probability (e.g., 99 %).

La medida desarrollada por Koza está aplicada a algoritmos genéticos, por esta razón se habla de "número de individuos a procesar". En términos más generales podríamos hablar de evaluaciones de candidatos o llamadas a la función objetivo. El proceso de cálculo es el siguiente:

a) Definimos la probabilidad de encontrar la solución a un problema condicionada al número de evaluaciones realizadas: $P(S|i)$, como la probabilidad acumulada entre 0 e i de encontrar la solución al problema.

   Esta probabilidad se estima experimentalmente.

b) La probabilidad $P(S|i)$ hace referencia a una sola ejecución del algoritmo. Por tanto, si el algoritmo lo ejecutamos R veces de manera independiente y en cada ejecución realizamos i evaluaciones, la probabilidad de encontrar la solución al problema *al menos una vez* en las R ejecuciones viene dada por:

$$1 - [1 - P(S|i)]^R$$

c) Si queremos garantizar la solución del problema con una probabilidad z, entonces:

$$z = 1 - [1 - P(S|i)]^R$$

   Utilizando la igualdad anterior, podemos obtener un valor de R en función de i:

$$R(i) = \frac{\log(1 - z)}{\log(1 - P(S|i))}$$

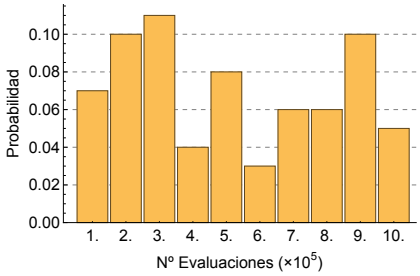
d) Una vez obtenido el valor de $R(i)$ sabemos que necesitamos ejecutar el algoritmo $R(i)$ veces. En cada ejecución evaluaremos i candidatos, con lo cual el esfuerzo computacional será: $R(i) \cdot i$.

Por ejemplo, supongamos que ejecutamos un algoritmo para obtener la solución a un problema. Al algoritmo le dejamos evaluar como mucho 10^6 candidatos y lo ejecutamos de manera independiente 100 veces.

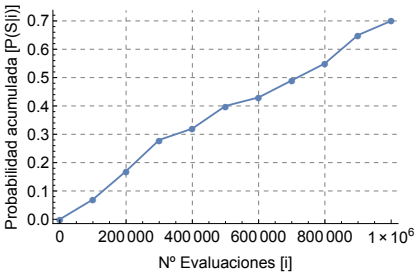
A partir de los datos empíricos, obtenemos un histograma de la probabilidad de encontrar la solución al problema: figura B.1a. En este ejemplo hemos realizado una agrupación utilizando un tamaño de 10^5 evaluaciones.

A partir del histograma de la probabilidad obtenemos la probabilidad acumulada ($P(S|i)$), figura B.1b. Con la probabilidad acumulada podemos calcular el número de ejecuciones independientes necesarias para resolver el problema con una seguridad del 99 % ($z = 0{,}99$), figura B.1c. Finalmente podemos obtener el esfuerzo computacional necesario en función del número de evaluaciones que hagamos por ejecución: figura B.1d.

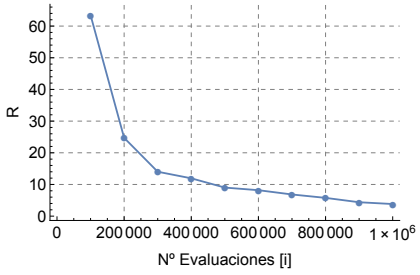
En este ejemplo, el esfuerzo computacional mínimo para resolver el problema con una probabilidad del 99 %, se obtiene al ejecutar el algoritmo 4 veces permitiéndole evaluar hasta 10^6 candidatos en cada ejecución.



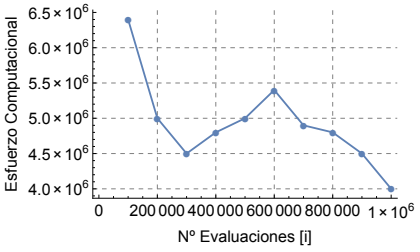
(a) Histograma de probabilidad.



(b) Probabilidad acumulada: $P(S|i)$.



(c) Valor de $R(i)$ para $z = 0{,}99$.



(d) Esfuerzo computacional.

Figura B.1: Ejemplo sencillo del cálculo del esfuerzo computacional.

Apéndice C

Representación y aprendizaje

Supongamos que queremos obtener una representación para que un computador sea capaz de aprender a jugar al ajedrez. En la figura C.1 se representa un movimiento muy simple de las blancas: comer un peón. Dicho movimiento lo podemos almacenar como conocimiento mediante una sencilla asociación estado-acción.

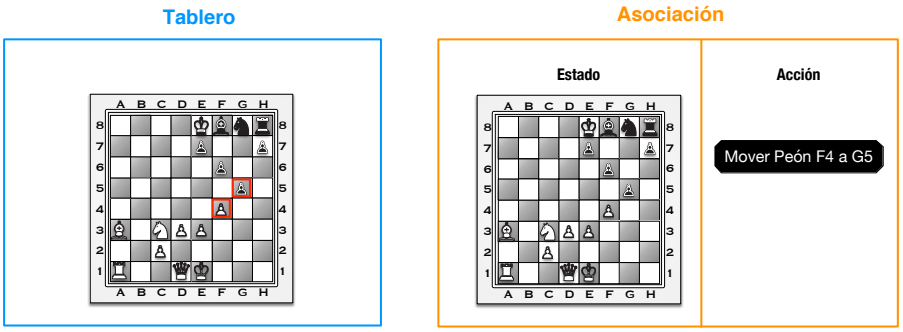


Figura C.1: Representación de la jugada de blancas utilizando una asociación estado-acción.

Esta representación plantea un problema, *el estado es tan especifico que dificulta reutilizar el conocimiento*. Supongamos la figura C.2, como se puede ver la situación del tablero es casi idéntica al estado contenido en la asociación, excepto por un peón que ocupa una posición diferente. Como los patrones no encajan no podemos aplicar el conocimiento adquirido de mover el peón blanco para comer el peón de de las negras.

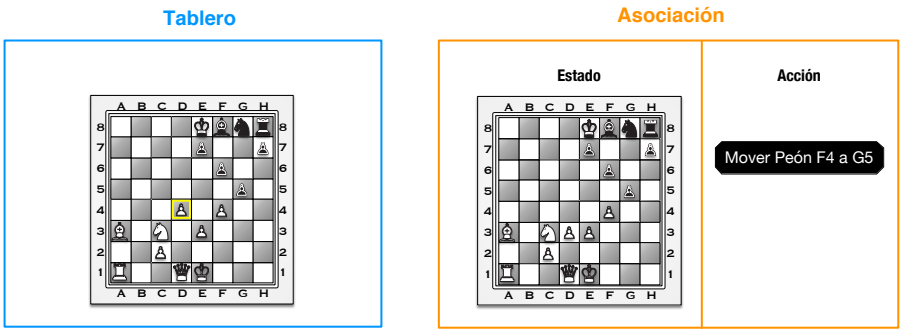


Figura C.2: Situación del tablero y el conocimiento disponible en forma de asociación.

Cuando estamos desarrollando una representación para permitir el aprendizaje donde el conocimiento se almacena en forma de asociación, es fundamental atender a la especificidad de los estados. Los estados representan patrones de una determinada situación, si dicho patrón es muy especifico –contiene mucha información– por ejemplo representa todo el tablero de ajedrez, será muy difícil aplicar la acción que asociemos con él. Si disminuimos la especificidad de los estados –la información que contienen–, será más fácil aplicar el conocimiento asociado a ellos. En el ejemplo que estamos utilizando podemos disminuir la información de los estados haciendo que no representan el tablero completo, únicamente una parte. Veamos un ejemplo.

Supongamos que definimos el estado como las ocho casillas adyacentes a una pieza. La representación de la misma jugada: comer un peón, sería la que se muestra en la figura C.3. Como el estado queda definido localmente al peón, la acción se almacena de forma relativa al estado porque hay que trasladar el movimiento a la posición del tablero que ocupa el peón (G4, H2, etc...). La virtud de esta representación es que nos permite aplicar el conocimiento de la jugada en muchas otras situaciones, por ejemplo en la mostrada en la figura C.4.

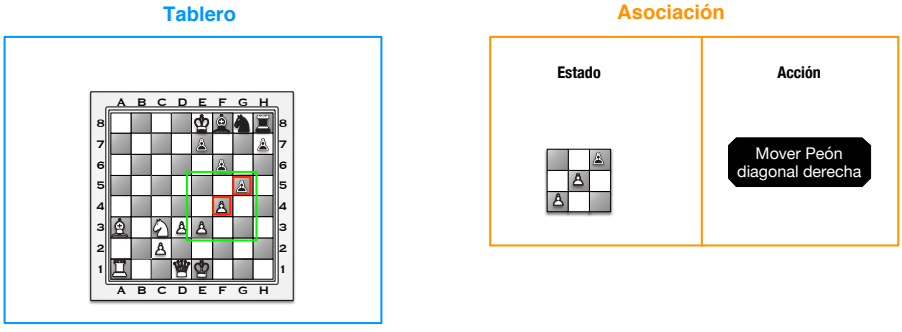


Figura C.3: Representación de la jugada de blancas definiendo el estado como las casillas adyacentes a una pieza.

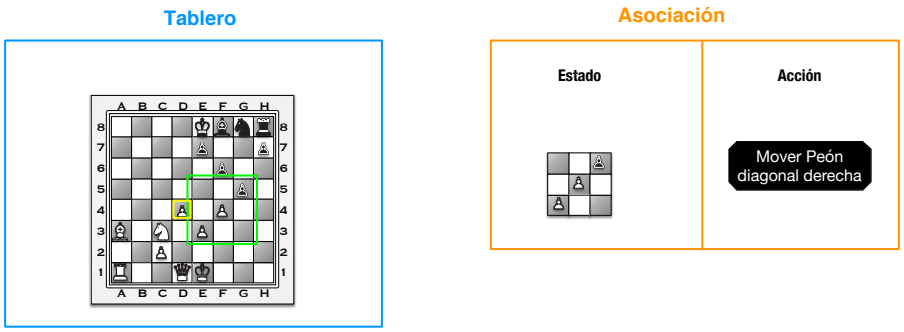


Figura C.4: Situación del tablero y el conocimiento disponible en forma de asociación cuando el estado está definido como las casillas adyacentes.

El uso de patrones locales, como las casillas adyacentes, para definir el estado nos permite aplicar fácilmente el conocimiento generado de una situación en otra situación distinta. De este modo podemos aplicar los movimientos aprendidos de una partida de ajedrez a otra distinta. Pero plantea un problema nuevo.

Cuando definimos el estado como el tablero entero, sólo existía un estado para cada posible situación del tablero. Al utilizar patrones locales, *una misma situación puede contener múltiples estados*. Por ejemplo de una situación del tablero podemos definir tantos estados como piezas tengamos.

Supongamos el tablero que se muestra en la figura C.5, aunque existen más estados únicamente consideramos tres en el ejemplo. Para aplicar el conocimiento indicado en la asociación debemos elegir el estado correcto: estado 3, de lo contrario el conocimiento no será aplicable.

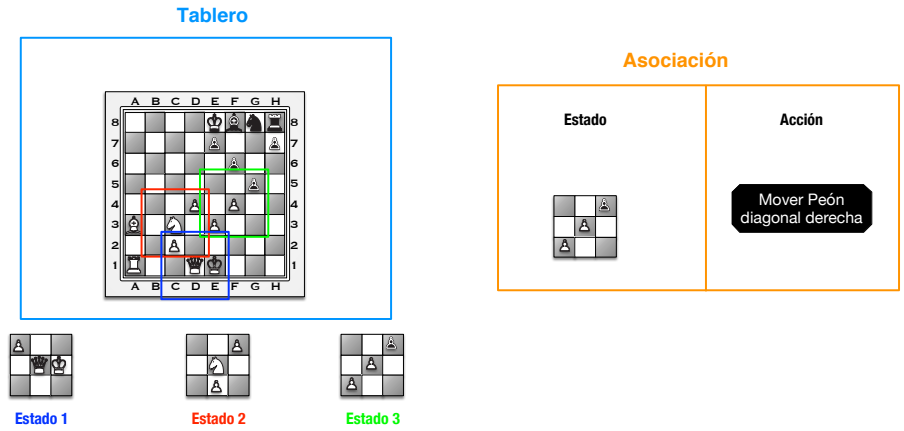


Figura C.5: Situación del tablero y posibles estados asociados.

Si queremos que el aprendizaje sea eficaz, la representación utilizando patrones locales debe incluir algún mecanismo que permita seleccionar el estado adecuado a cada situación. Una forma posible es incluir la selección del estado (del patrón local) en el propio conocimiento que se almacena. Es decir, dado un patrón local permitir la aplicación de una acción que cambie el patrón local pero

sin alterar la situación global. En nuestro ejemplo del ajedrez consistiría en una acción que nos permita "fijarnos" en otra pieza pero sin mover ninguna, de este modo cambiamos de estado (cambiamos de patrón) pero el conjunto de estados posibles asociados a la situación del tablero permanece inalterado.

Por ejemplo, si observamos la figura C.6, toda la jugada de comer un peón de negras se realiza utilizando el conocimiento asociado. Partimos del estado determinado por la reina. En base al conocimiento disponible, no interesa mover la reina, por tanto la acción consiste en seleccionar el caballo. Igualmente no interesa mover el caballo, pasamos al patrón del peón. Una vez en el patrón del peón, realizamos el movimiento de la pieza y termina nuestra jugada.

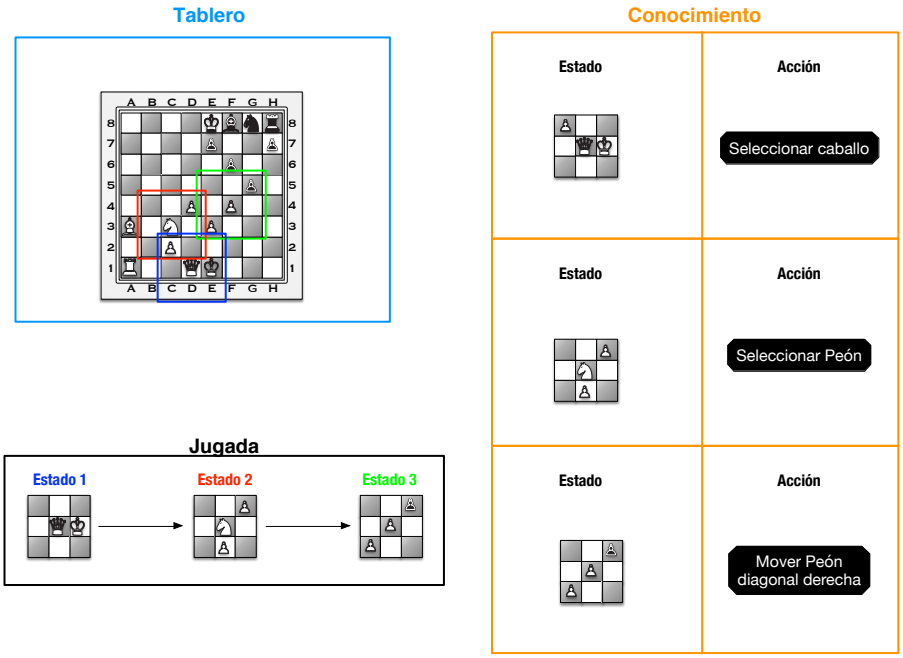


Figura C.6: Situación del tablero y posibles estados asociados.

La idea es ir procesando los estados sucesivamente: una pieza después de otra, por lo tanto es importante advertir lo siguiente:

i) Mientras la situación global no cambie (el tablero sea el mismo), es importante evitar que aparezcan transiciones infinitas, por ejemplo: seleccionar reina, seleccionar caballo, seleccionar reina, seleccionar caballo, etc...

ii) Cuando la situación global (el tablero) cambia, por ejemplo movemos una pieza. Debemos volver permitir visitar todos los estados otra vez, es decir, poder comprobar todas las piezas.

Una forma sencilla de resolver este problema es definir un estado inicial, por ejemplo el rey, a partir del cual avanzamos sin repetir piezas. Cuando movemos una pieza, simplemente volvemos al estado inicial. De este modo cuando el contrincante mueve y es nuestro turno otra vez, podemos acceder a todas nuestras piezas de nuevo.

Utilizando esta aproximación es posible obtener una representación que permita a un algoritmo aprender de una partida a otra. No estamos hablando de conseguir un campeón, sino de algo más básico como aprender a mover la reina si se encuentra amenazada, en vez de mover un peón y perder la reina. Es decir, a medida que el computador juegue más partidas poder observar un comportamiento inteligente consistente en ir aprendiendo una serie de reglas básicas.

Funciones de similitud. El aprendizaje debe entenderse como resolver situaciones nuevas haciendo uso del conocimiento obtenido de situaciones pasadas que comparten algún rasgo con la actual. Más concretamente aplicar el conocimiento que tenemos acerca de como actuar en un determinado estado a multitud de ellos que consideremos "parecidos".

Una solución muy extendida para tratar con multitud de estados es utilizar *funciones de similitud*. Estas funciones sirven para clasificar los estados en grupos de acuerdo con algún criterio de similitud. El conocimiento simplemente hace referencia a tipos de estados: grupos de estados similares.

En ocasiones ocurre que no es fácil definir una función de similitud. Por ejemplo, el caballo negro amenaza a nuestra reina. Por un lado la amenaza a nuestra reina podría ser un componente similar a otras partidas, pero otras piezas pueden estar dispuestas de manera completamente distinta. Se puede ver la problemática: ¿es un estado similar?, ¿podemos aplicar el conocimiento almacenado acerca de cómo proteger a nuestra reina?.

Si utilizamos patrones locales la información relevante a nuestra reina es local a ella: sus casillas adyacentes. Por tanto, aunque las partidas sean distintas, la reina se puede ver amenazada por el mismo patrón *local* y sabremos como defenderla.

En vez de definir reglas explícitas para la similitud de estados, restringimos la información que define cada estado. Implícitamente, la similitud surge –emerge– del número de patrones que comparten situaciones diferentes. De este modo, no es necesario definir ninguna comparación explícita, sino que basta con vincular la acción al patrón local. El propio sistema aprende qué patrones son relevantes para actuar –mover una pieza– y cuales conviene ignorar –seleccionar otra pieza–. Al final, cuando el sistema adquiere el suficiente conocimiento es capaz de desarrollar una conducta inteligente.

Apéndice D

Esquema de la implementación

En este apéndice se agrupan todos los fragmentos de código mostrados a lo largo del documento para dar una visión completa de cómo resulta finalmente la implementación de ACE.

```
1: μ ← ∞                                                    ▷ Inicialización
2: procedure Bucle Principal
3:     for a ∈ Población do                                  ▷ Paso de búsqueda
4:         {L_U, P(L_U)} ← política de control
5:         acción ← selección(L_U, P(L_U))
6:         agente ← actualizar(agente)
7:     end for
8:     for a ∈ Población do                                  ▷ Actualización de información
9:         if finalizado(a) == ⊤ then
10:            S ← a(S)                                     ▷ Obtenemos la solución a partir del agente
11:            if J(S) < μ then                             ▷ Criterio de umbral
12:                if J(S) < J_best then                    ▷ Actualizamos la mejor solución
13:                    J_best ← J(S)
14:                end if
15:                μ ← actualizaMedia(S, m)
16:                λ ← (J(s)−μ)/(J_best−μ)                  ▷ Calidad de la solución (Eq. 3.6).
17:                τ ← actualizaTabla(a(A_q), a(A_u), λ)
18:                exito(a)                                 ▷ Dinámica de población
19:            else
20:                fracaso(a)                               ▷ Dinámica de población
21:            end if
22:        end if
23:    end for
24:    reclutamiento                                        ▷ Dinámica de población
25: end procedure
26:
```

Figura D.1: Bucle principal del algoritmo.

Tanto la población de agentes, como la tabla de feromona comienzan siendo estructuras vacías. El propio proceso de búsqueda se encarga de irlas generando. Excepto la media, el resto variables se inicializan a cero

```
1:  procedure actualizaMedia(S, m)
2:      m ← m + 1
3:      L_μ ← add(J(S))
4:      if m = 1 then
5:          μ ← J(S)                          ▷ Inicialización de la media
6:      else
7:          if m < γ_3 then
8:              μ ← μ + (J(S) − μ)/m
9:          else
10:             μ ← μ + J(S)/γ_3 − L_μ(γ_3)/γ_3
11:             L_μ ← extraer(L_μ(γ_3))
12:             m ← m − 1
13:         end if
14:     end if
15: end procedure
```

Figura D.2: Procedimiento para actualizar el valor de la media.

```
1:  procedure actualizaTabla(A_q, A_u, λ)
2:      for i ∈ |A_q| do                      ▷ Estados visitados por el agente
3:          q_i ← A_q(i)
4:          L_τ ← τ(q_i)          ▷ Contenido de la tabla para el estado q_i
5:          for j ∈ |L_τ| do
6:              [τ_j, u_j] ← L_τ(j)
7:              if u_j == A_u(i) then
8:                  τ_j = τ_j + (1 − τ_j) · λ
9:              else
10:                 τ_j = τ_j + (0 − τ_j) · λ
11:             end if
12:         end for
13:     end for
14: end procedure
```

Figura D.3: Procedimiento para actualizar el contenido de la tabla de feromona.

```
1: procedure Éxito(ant)
2:     if ant es Forager then
3:         FR ← FR − 1
4:         SP ← 0
5:         RP ← 1
6:     else
7:         PR ← PR − 1
8:         SP ← SP + 1
9:         UP ← 0
10:        if SP ≥ FR then
11:            RF ← 1
12:        else
13:            RP ← 1
14:        end if
15:    end if
16: end procedure
```

Figura D.4: Procedimiento para actualizar las variables de control de la dinámica de población en caso de éxito.

```
1: procedure Fracaso(ant)
2:     if ant es Forager then
3:         FR ← FR − 1
4:         if (PR + FR) · (1 − J_best/μ) > FR then
5:             RF ← 1
6:         else
7:             RP ← 1
8:         end if
9:     else
10:        PR ← PR − 1
11:        UP ← UP + 1
12:        RP ← round(log(UP + 1))
13:        if (PR + FR) · (1 − J_best/μ) > FR then
14:            RF ← 1
15:        end if
16:    end if
17: end procedure
```

Figura D.5: Procedimiento para actualizar las variables de control de la dinámica de población en caso de fallo.

```
1: procedure RECLUTAMIENTO
2:     if m < 1 then                                  ▷ Fase de inicialización
3:         Población ← activarPatroller()
4:         PR ← PR + 1
5:     else
6:         while RP > 0 do
7:             Población ← activarPatroller()
8:             PR ← PR + 1
9:             RP ← RP − 1
10:        end while
11:        while RF > 0 do
12:            Población ← activarForager()
13:            FR ← FR + 1
14:            RF ← RF − 1
15:        end while
16:    end if
17: end procedure
```

Figura D.6: Procedimiento para actualizar la población.

Bibliografía

[AC07] Enrique Alba and Francisco Chicano. Acohg: Dealing with huge graphs. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 10–17. ACM, 2007.

[AHM02] Hussein A Abbass, Xuan Hoai, and Robert I Mckay. Anttag: A new method to compose computer programs using colonies of ants. In *Computational Intelligence, Proceedings of the World on Congress on*, volume 2, pages 1654–1659. IEEE, 2002.

[Axe97] Robert M Axelrod. *The complexity of cooperation: Agent-based models of competition and collaboration*. Princeton University Press, 1997.

[BDT99] Eric Bonabeau, Marco Dorigo, and Guy Theraulaz. *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press, USA, 1 edition, 1999.

[Blu02] Christian Blum. Aco applied to group shop scheduling: A case study on intensification and diversification. In *Ant algorithms*, pages 14–27. Springer, 2002.

[BM03] J. Marcos Moreno Vega Belén Melían, José A. Moreno Pérez. Metaheurísticas: Una visión global. *Inteligencia Artificial, Revista Iberoamericana de Inteligencia Artificial.*, (19):7–28, 2003.

[BN99] Franz Baader and Tobias Nipkow. *Term rewriting and all that*. Cambridge University Press, 1999.

[BR03] Christian Blum and Andrea Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys (CSUR)*, 35(3):268–308, 2003.

[Bro90] Rodney A Brooks. Elephants don't play chess. *Robotics and autonomous systems*, 6(1):3–15, 1990.

[Bro91] Rodney A Brooks. Intelligence without representation. *Artificial intelligence*, 47(1):139–159, 1991.

[Bro99] Rodney Allen Brooks. *Cambrian intelligence: the early history of the new AI*. Mit Press, 1999.

[BTD96] E. Bonabeau, G. Theraulaz, and J.L. Deneubourg. Quantitative study of the fixed threshold model for the regulation of division of labour in

insect societies. *Proceedings: Biological Sciences*, 263(1376):1565–1569, 1996.

[BW93] Gerardo Beni and Jing Wang. Swarm intelligence in cellular robotic systems. In *Robots and Biological Systems: Towards a New Bionics?*, pages 703–712. Springer, 1993.

[CDF+03] Scott Camazine, Jean-Louis Deneubourg, Nigel R Franks, James Sneyd, Guy Theraula, and Eric Bonabeau. *Self-Organization in Biological Systems:(Princeton Studies in Complexity)*. Princeton University Press, 2003.

[CMB12] B Chandra Mohan and R Baskaran. A survey: Ant colony optimization based recent research and implementation on several engineering domain. *Expert Systems with Applications*, 39(4):4618–4627, 2012.

[CO02] Steffen Christensen and Franz Oppacher. An analysis of koza's computational effort statistic for genetic programming. In *Genetic programming*, pages 182–191. Springer, 2002.

[CO07] Steffen Christensen and Franz Oppacher. Solving the artificial ant on the santa fe trail problem in 20,696 fitness evaluations. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 1574–1579. ACM, 2007.

[DA90] René Descartes and Guillermo Quintás Alonso. *El tratado del hombre*. Alianza Editorial, 1990.

[DCD97] Gianni Di Caro and Marco Dorigo. Antnet: A mobile agents approach to adaptive routing. Technical report, Technical Report IRIDIA/97-12, IRIDIA, Université Libre de Bruxelles, Belgium, 1997.

[DG97] M. Dorigo and L.M. Gambardella. Ant colony system: A cooperative learning approach to the traveling salesman problem. *Evolutionary Computation, IEEE Transactions on*, 1(1):53–66, 1997.

[DMC96] Marco Dorigo, Vittorio Maniezzo, and Alberto Colorni. Ant system: optimization by a colony of cooperating agents. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 26(1):29–41, 1996.

[DMRP10] Yves Demazeau, Jörg P Müller, Juan M Corchado Rodríguez, and Javier Bajo Pérez. *Advances in Practical Applications of Agents and Multiagent Systems*. Springer, 2010.

[DS04] Marco Dorigo and Thomas Stützle. *Ant Colony Optimization (Bradford Books)*. The MIT Press, June 2004.

[EJG09] Jose B. Escario, Juan F. Jiménez, and Jose M. Giron-Sierra. Ant colony optimized planning for unmanned surface marine vehicles. In *Instrumentation viewpoint*, volume 8, 2009.

[EJG10a] Jose B. Escario, Juan F. Jiménez, and Jose M. Giron-Sierra. Ant colony extended: Search in solution spaces with a countably infinite number of solutions. In *Swarm Intelligence - 7th International Conference, ANTS 2010, Brussels, Belgium, September 8-10, 2010. Proceedings*, pages 552–553, 2010.

[EJG10b] Jose B. Escario, Juan F. Jiménez, and Jose M. Giron-Sierra. Optimization of autonomous ship maneuvers applying swarm intelligence. In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, Istanbul, Turkey, 10-13 October 2010*, pages 2603–2610, 2010.

[EJGS11] J.B. Escario, J.F. Jimenez, and J.M. Giron-Sierra. Ship maneuvering planning using swarm intelligence. In *OCEANS, 2011 IEEE - Spain*, pages 1–9, 2011.

[EJGS12] Jose B Escario, Juan F Jimenez, and Jose M Giron-Sierra. Optimisation of autonomous ship manoeuvres applying ant colony optimisation metaheuristic. *Expert Systems with Applications*, 39(11):10120–10139, 2012.

[EJGS13] Jose B. Escario, Juan F. Jimenez, and Jose M. Giron-Sierra. *Self-organization: Theories and Methods*, chapter Self-organization and Task Allocation: An Application to Ant Algorithms (Chapter 3). Nova Science Publisher In., New York, 2013.

[EJGS15] Jose B Escario, Juan F Jimenez, and Jose M Giron-Sierra. Ant colony extended: Experiments on the travelling salesman problem. *Expert Systems with Applications*, 42(1):390–410, 2015.

[Fla98] Gary William Flake. *The computational beauty of nature: Computer explorations of fractals, chaos, complex systems, and adaptation*. MIT press, 1998.

[For90] Stephanie Forrest. Emergent computation: An introduction. *Physica D: Nonlinear Phenomena*, 42(1):1–11, 1990.

[For91] Stephanie Forrest. *Emergent computation: self-organizing, collective, and cooperative phenomena in natural and artificial computing networks*. Special issues of physica D. MIT Press, 1991.

[GADP89] Simon Goss, Serge Aron, Jean-Louis Deneubourg, and Jacques Marie Pasteels. Self-organized shortcuts in the argentine ant. *Naturwissenschaften*, 76(12):579–581, 1989.

[GC95] Nigel Gilbert and Rosaria Conte. *Artificial Societies: the computer simulation of social life*. Taylor & Francis, Inc., 1995.

[GC03] Jean Dickinson Gibbons and Subhabrata Chakraborti. *Nonparametric statistical inference*. CRC press, 2003.

[GD94] Nigel Gilbert and Jim Doran. Simulating societies. the computer simulation of social phenomena. 1994.

[Gha11] Jamshid Gharajedaghi. *Systems thinking: Managing chaos and complexity: A platform for designing business architecture*. Elsevier, 2011.

[Gil95] Nigel Gilbert. Emergence in social simulation. *Artificial societies: The computer simulation of social life*, pages 144–156, 1995.

[GK96] Deborah M Gordon and Alan W Kulig. Founding, foraging, and fighting: colony size and the spatial distribution of harvester ant nests. *Ecology*, pages 2393–2409, 1996.

[GK03] Fred Glover and Gary A Kochenberger. *Handbook of metaheuristics*. Springer, 2003.

[Glo86] Fred Glover. Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, 13(5):533–549, 1986.

[Gol06] David E Goldberg. *Genetic algorithms*. Pearson Education India, 2006.

[Gor89] Deborah M Gordon. Dynamics of task switching in harvester ants. *Animal Behaviour*, 38(2):194–204, 1989.

[Gor91] Deborah M Gordon. Behavioral flexibility and the foraging ecology of seed-eating ants. *American Naturalist*, pages 379–411, 1991.

[Gor96] Deborah M Gordon. The organization of work in social insect colonies. *Nature*, 380(6570):121–124, 1996.

[Gor99] Deborah M Gordon. *Ants at work: how an insect society is organized*. Simon and Schuster, 1999.

[Gor10] Deborah M Gordon. *Ant encounters: interaction networks and colony behavior*. Princeton University Press, 2010.

[Gra59] Plerre-P Grassé. La reconstruction du nid et les coordinations interindividuelles chezbellicositermes natalensis etcubitermes sp. la théorie de la stigmergie: Essai d'interprétation du comportement des termites constructeurs. *Insectes sociaux*, 6(1):41–80, 1959.

[Her98] R. Hermida. *Fundamentos de computadores*. Manuales científico-técnicos. Síntesis, 1998.

[Hol75] John H Holland. *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence*. U Michigan Press, 1975.

[Hop07] John E Hopcroft. *Introduction to automata theory, languages, and computation*. Pearson Addison Wesley, 2007.

[HS04] H.H. Hoos and T. Stützle. *Stochastic Local Search: Foundations & Applications*. The Morgan Kaufmann Series in Artificial Intelligence. Elsevier Science, 2004.

[HW90] B. Hölldobler and E.O. Wilson. *The Ants*. Belknap Press of Harvard University Press, 1990.

[HW09] Bert Hölldobler and Edward O Wilson. *The superorganism: the beauty, elegance, and strangeness of insect societies*. WW Norton & Company, 2009.

[JM97] D.S. Johnson and L.A. McGeoch. The traveling salesman problem: A case study in local optimization. *Local search in combinatorial optimization*, pages 215–310, 1997.

[Joh02] Steven Johnson. *Emergence: The connected lives of ants, brains, cities, and software*. Simon and Schuster, 2002.

[Kau93] Stuart A. Kauffman. *The origins of order: Self-organization and selection in evolution*. Oxford university press, 1993.

[Kau95] Stuart Kauffman. *At home in the universe: The search for the laws of self-organization and complexity*. Oxford University Press, 1995.

[KGV+83] Scott Kirkpatrick, C Daniel Gelatt, Mario P Vecchi, et al. Optimization by simmulated annealing. *science*, 220(4598):671–680, 1983.

[KOKG12] Dervis Karaboga, Celal Ozturk, Nurhan Karaboga, and Beyza Gorkemli. Artificial bee colony programming for symbolic regression. *Information Sciences*, 209:1–15, 2012.

[Koz92] J.R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection (Complex Adaptive Systems)*. A Bradford Book, 1992.

[KS02] Christian Keber and Matthias G Schuster. Option valuation with generalized ant programming. In *GECCO*, pages 74–81, 2002.

[L+89] Christopher G Langton et al. *Artificial life*. Addison-Wesley Publishing Company Redwood City, CA, 1989.

[Lan90] Chris G Langton. Computation at the edge of chaos: phase transitions and emergent computation. *Physica D: Nonlinear Phenomena*, 42(1):12–37, 1990.

[LHL05] Jason D Lohn, Gregory S Hornby, and Derek S Linden. An evolved antenna for deployment on nasa's space technology 5 mission. In *Genetic Programming Theory and Practice II*, pages 301–315. Springer, 2005.

[LP02] William B Langdon and Riccardo Poli. *Foundations of genetic programming*. Springer, 2002.

[LP06] Sean Luke and Liviu Panait. A comparison of bloat control methods for genetic programming. *Evolutionary Computation*, 14(3):309–344, 2006.

[LPB+06] Sean Luke, Liviu Panait, Gabriel Balan, Sean Paus, Zbigniew Skolicki, Jeff Bassett, Robert Hubley, and A Chircop. Ecj: A java-based evolutionary computation research system, 2006.

[McC56] John McCarthy. The inversion of functions defined by turing machines. *Automata studies*, pages 177–181, 1956.

[Mea08] Donella H Meadows. *Thinking in systems: A primer*. Chelsea Green Publishing, 2008.

[Min88] Marvin Minsky. *Society of mind*. Simon and Schuster, 1988.

[Mit09] Melanie Mitchell. *Complexity: A guided tour*. Oxford University Press, 2009.

[MJE13] R. Mattila, J. Jimenez, and J.B. Escario. Including bathymetric data in autonomous surface vessels' maneuvering optimisation tool. Technical report, Universidad Complutense de Madrid, 2013.

[MP09] John H Miller and Scott E Page. *Complex Adaptive Systems: An Introduction to Computational Models of Social Life: An Introduction to Computational Models of Social Life*. Princeton University Press, 2009.

[NS+72] Allen Newell, Herbert Alexander Simon, et al. *Human problem solving*, volume 104. Prentice-Hall Englewood Cliffs, NJ, 1972.

[Pol03] Riccardo Poli. A simple but theoretically-motivated method to control bloat in genetic programming. In *Genetic Programming*, pages 204–217. Springer, 2003.

[Rei94] Gerhard Reinelt. *The traveling salesman: computational solutions for TSP applications*. Springer-Verlag, 1994.

[Rei95] Gerhard Reinelt. Tsplib 95, 1995.

[Rey99] Craig W Reynolds. Steering behaviors for autonomous characters. In *Game developers conference*, volume 1999, pages 763–782, 1999.

[Rey00] Craig W Reynolds. Interaction with groups of autonomous characters. In *Game Developers Conference*, volume 2000, page 83, 2000.

[RN95] Stuart Jonathan Russell and Peter Norvig. *Artificial intelligence: a modern approach*. Prentice hall Englewood Cliffs, 1995.

[SAW08] Amirali Salehi-Abari and Tony White. Enhanced generalized ant programming (egap). In *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pages 111–118. ACM, 2008.

[SCS91] Thomas D Seeley, Scott Camazine, and James Sneyd. Collective decision-making in honey bees: how colonies choose among nectar sources. *Behavioral Ecology and Sociobiology*, 28(4):277–290, 1991.

[SG08] Ricard Solé and Brian Goodwin. *Signs of life: How complexity pervades biology*. Basic books, 2008.

[SH00] Thomas Stützle and Holger H. Hoos. MAX-MIN Ant System. *Future Generation Computer Systems*, 16(8):889–914, 2000.

[Sha48] C.E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27(3):379–423, 1948.

[Sim69] Herbert Alexander Simon. *The sciences of the artificial*, volume 136. MIT press, 1969.

[Sip06] Michael Sipser. *Introduction to the Theory of Computation*. Cengage Learning, 2006.

[SLB09] Yoav Shoham and Kevin Leyton-Brown. *Multiagent systems: Algorithmic, game-theoretic, and logical foundations*. Cambridge University Press, 2009.

[Sny05] Jan Snyman. *Practical mathematical optimization: an introduction to basic optimization theory and classical and new gradient-based algorithms*, volume 97. Springer, 2005.

[Stu96] T Stutzle. Improving the ant system: A detailed report on the max-min ant system. Technical report, 1996.

[Stu97] Stutzle. MAX-MIN Ant System and local search for the traveling salesman problem. In *Evolutionary Computation, 1997., IEEE International Conference on*, pages 309–314, 1997.

[Stü98] Thomas Stützle. An ant approach to the flow shop problem. In *Proceedings of the 6th European Congress on Intelligent Techniques and Soft Computing (EUFIT98)*, volume 3, pages 1560–1564, 1998.

[Tur50] Alan M Turing. Computing machinery and intelligence. *Mind*, pages 433–460, 1950.

[VF67] Karl Von Frisch. The dance language and orientation of bees. 1967.

[Wat99] Duncan J Watts. *Small worlds: the dynamics of networks between order and randomness*. Princeton university press, 1999.

[Wei99] Gerhard Weiss. *Multiagent systems: a modern approach to distributed artificial intelligence*. MIT press, 1999.

[Wil00] Edward O Wilson. *Sociobiology: The new synthesis*. Harvard University Press, 2000.

[WJK00] Michael Wooldridge, Nicholas R Jennings, and David Kinny. The gaia methodology for agent-oriented analysis and design. *Autonomous Agents and Multi-Agent Systems*, 3(3):285–312, 2000.

[Wol02] Stephen Wolfram. *A new kind of science*, volume 5. Wolfram media Champaign, 2002.

[ZJW03] Franco Zambonelli, Nicholas R Jennings, and Michael Wooldridge. Developing multiagent systems: The gaia methodology. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 12(3):317–370, 2003.

Lista de Figuras

Lista de publicaciones asociadas a la tesis

Publicaciones en revistas internacionales

Título: Ant Colony Extended: Experiments on the Travelling Salesman Problem

Autores: José B. Escario, Juan F. Jiménez, José M. Girón-Sierra

Congreso: Expert System with Applications

Año: 2015

Título: Optimisation of autonomous ship manoeuvres applying Ant Colony Optimisation metaheuristic

Autores: José B. Escario, Juan F. Jiménez, José M. Girón-Sierra

Congreso: Expert System with Applications

Año: 2012

Capítulos de libros

Título: Self-organization: Theories and Methods

Capítulo: Self-organization and Task Allocation: An Application to Ant Algorithms

Autores: José B. Escario, Juan F. Jiménez, José M. Girón-Sierra

Editorial: Nova Science Publisher

Año: 2013

Publicaciones en congresos internacionales

Título: Ship maneuvering planning using swarm intelligence

Autores: José B. Escario, Juan F. Jiménez, José M. Girón-Sierra

Congreso: Oceans 2011 IEEE

Año: 2011

Título: Optimization of autonomous ship maneuvers applying swarm intelligence

Autores: José B. Escario, Juan F. Jiménez, José M. Girón-Sierra

Congreso: IEEE International Conference on Systems, Man and Cybernetics

Año: 2010

Título: Ant Colony Extended: Search in Solution Spaces with a Countably Infinite Number of Solutions

Autores: José B. Escario, Juan F. Jiménez, José M. Girón-Sierra

Congreso: Swarm Intelligence - 7th International Conference, (ANTS 2010)

Año: 2010

Título: Ant colony optimized planning for unmanned surface marine vehicles

Autores: José B. Escario, Juan F. Jiménez, José M. Girón-Sierra

Congreso: Instrumentation Viewpoint, n. 8

Título: Autonomous Ship Manoeuvring Planning Based on the Ant Colony Optimization Algorithm

Autores: José B. Escario, Juan F. Jiménez, José M. Girón-Sierra

Congreso: 8th IFAC Conference on Manoeuvring and Control of Marine Craft

Año: 2009

Publicaciones en otros medios

Título: Optimización de maniobras para barcos autónomos

Autores: José B. Escario, Juan F. Jiménez, José M. Girón-Sierra

Medio: Oficina de Transferencia de Resultados de Investigación (OTRI), Universidad Complutense de Madrid

Año: 2012

Glosario

Q Conjunto de estados.

q_0 Estado inicial.

Q_0 Conjunto de estados iniciales.

q Estado.

U Conjunto de acciones..

u Acción.

δ Función de transición.

J Función de coste.

A Agente.

A_Q Lista de estados recorridos por un agente.

A_U Lista de acciones aplicadas por un agente.

η Heurística: distribución de pesos que representa la Información previa.

τ Feromona: distribución de pesos que representa la Información adquirida.

μ Media de las soluciones descubiertas hasta el momento, se utiliza como valor umbral para determinar si una búsqueda tiene éxito o no.

Υ Función dependiente de un problema que indica cuales son las acciones disponibles para un estado dado.

γ_1 Parámetro definido por el usuario que regula el uso de la heurística por parte de los patrollers.

γ_2 Parámetro definido por el usuario que regula el uso de la heurística por parte de los patrollers.

γ_3 Parámetro definido por el usuario que regula el número de muestras para el calculo de la media móvil.

NS número de elementos contenidos en una secuencia que es solución de un problema.

λ valor entre 0 y 1 que representa la calidad asociada a una solución del problema.

v Frecuencia relativa del uso de una acción.

UP número de patrollers sin éxito desde el último patroller con éxito.

SP número de patrollers con éxito desde el último forager con éxito.

Siglas

DAI Inteligencia Artificial Distribuida.

ACO Ant Colony Optimisation.

AS Ant System.

TSP Travelling Salesman Problem.

ACE Ant Colony Extended.

ACS Ant Colony System.

MMAS Max-Min Ant System.

PSO Particle Swarm Optimization.